Mémoire de fin d'études

M2

Nom : METRO
Prénom : Benjamin
ID Booster : 58861
Campus : Océan Indien

Logiciel d'aide à la décision pour la culture d'ananas

Remerciements

Pour commencer, je tiens à remercier M. Patrick FOURNIER, pour nous avoir accepté Mickaël FOLIO et moi pour des stages à temps partiel et à temps plein, ces deux dernières années.
Au sein du Centre de Coopération Internationale en Recherche Agronomique pour le Développement (CIRAD), nous avons pu travailler sur le projet « Oumpapa », renommé par la suite « AnaGmaX », portant sur un domaine que nous ne connaissions pas malgré sa forte présence locale : la culture d'ananas.
Je remercie aussi toute l'équipe de la station de Bassin Plat, ainsi que les autres stagiaires du CIRAD, pour les aides, conseils, ainsi que pour l'accueil chaleureux.
Pour terminer je remercie mes camarades de classe, qui m'ont permis de passer de très agréables années au sein de l'école SUPINFO, à la fois par la bonne humeur que dégage la communauté de l'école, ainsi que par le soutien que les différents élèves s'apportent mutuellement, Et ce même pendant le stage, où la distance n'a pas changé ces liens que SUPINFO a créés.

Résumé

Mon rôle dans le projet Oumpapa consistait au début à réécrire l'application existante, écrite en Visual Basic sous Microsoft Access, en une application exécutable (.EXE) en C#.NET.
L'application existante rassemblait une grande partie des caractéristiques des parcelles d'ananas des utilisateurs (producteurs), et permettait de manière organisée de consulter l'évolution de la croissance, ainsi que la prévision des récoltes. L'application a été développée avec du code procédural, il a donc fallu réétudier l'application, depuis les données qu'elle utilise jusqu'aux moindres actions qu'elle exécute, et de réécrire la nouvelle application en orienté-objet, afin qu'elle puisse être par la suite modifiable et facilement évolutive.
Finalement le projet a été de réécrire l'application, mais en y apportant de profonds changements :
de la façon de voir les caractéristiques, de les quantifier, des les stocker jusqu'à leur traitement, C'est une toute nouvelle approche du problème que nous avons construite, Patrick, Mickaël et moi.
Ce mémoire présente les points qui m'ont permis de mener à bien ce projet depuis mon stage à temps partiel jusqu'à mon stage de fin d'étude.

Dans la première partie de ce mémoire, nous allons présenter les différents aspects du traitement des données concernant la culture d'ananas, à travers la présentation de la société.
Dans la deuxième partie, nous présenterons la manière dont nous avons conçu le nouveau logiciel.
Pour terminer, nous allons voir les bonnes méthodes à adopter pour qu'un programmeur puisse facilement exploiter et reprendre le travail d'un collègue.

Table des matières

Introduction

De nos jours, les systèmes de traitement des données sont présents dans de nombreux domaines, et ne se limitent pas aux entreprises ou aux activités purement informatiques. L'agriculture en fait partie, pour le moins qu'on s'y attende.

Le pôle de recherche agronomique du CIRAD a ainsi développé une application qui permet à l'agriculteur de gérer sa culture d'ananas. Cette application permet le suivi de la croissance, les statistiques, l'établissement de listes de taches ainsi que la prévision des récoltes et des rendements.

L'objectif du logiciel n'est pas de remplacer l'homme, mais bien de lui donner des outils pour mieux comprendre et mieux gérer sa plantation, pour optimiser sa production d'ananas, et ainsi mieux satisfaire sa clientèle (en terme de coût, de délais, de quantités, de disponibilité...).

Présentation de la société

La société

Tous mes stages (temps partiel, temps plein et stage de fin d'études) de ces deux dernières années à SUPINFO ont été réalisés au CIRAD (ou encore « Centre de Coopération Internationale en Recherche Agronomique pour le Développement »). Ce centre français est un Établissement public à caractère industriel et commercial (EPIC), statut qui est utilisé pour désigner une société publique représentée par une personne publique, et ayant pour but la gestion d'une activité de service public. Ce statut a été créé pour faire face à un besoin qui pourrait être assuré par une entreprise industrielle ou commerciale, mais qui, compte tenu des circonstances, ne peut pas être correctement effectué par une entreprise privée soumise à la concurrence. Ce concept est remis en cause par la pensée libérale, qui tend à privatiser au maximum les entreprises. Les EPIC sont la plupart du temps régies par les lois privées même si, en tant qu'entités publiques légales, elle bénéficient de quelques privilèges publics.

Le dispositif du CIRAD basé à la Réunion est le plus grand en dehors de l'Hexagone. Elle œuvre dans le but de proposer des solutions innovantes aux problèmes que rencontrent l'agriculture dans sa région pour se développer durablement, et afin de faire progresser la connaissance dans les différents domaines scientifiques.

Chiffres clés

Le budget du CIRAD en 2008 s'élevait à 203 millions d'euros. La société emploie 1800 agents, dont 800 chercheurs. Ce personnel est réparti sur 3 départements scientifiques, en 46 unités de recherche distinctes, et 7 pôles scientifiques à vocation régionale dans l'outre-mer. Il dispose d'un réseau mondial de partenaires et de 12 directions régionales, à partir desquelles il mène des activités de coopération avec plus de 90 pays.

Fonctions, structure et organisation

Fonctions dans la société

A la Réunion, le CIRAD est structuré, depuis le 1er janvier 2007, en trois pôles de recherche :

- Qualité des productions agricoles et alimentaires tropicales (pôle KAPPA),
- Risque environnemental, agriculture et gestion intégrée des ressources (pôle REAGIR),
- Protection des plantes (3P)

Nous allons voir plus en détail le rôle du département dans lequel j'ai effectué mes stages, le pôle KAPPA :

La qualité d'un produit est le résultat d'un long processus qui débute chez le producteur, se poursuit lors de la transformation et du stockage, pour être finalement notée en fin de chaîne par le consommateur. Pour l'appréhender, il est nécessaire d'adopter une approche intégrée, de la "fourche à la fourchette", qui mesure la qualité, de l'exploitation agricole jusqu'à l'assiette du consommateur.

Le pôle KAPPA a pour objectif majeur de construire des itinéraires techniques, du producteur au consommateur, garantissant la traçabilité et la qualité des produits, aussi bien sous ses aspects sanitaires, sensoriels que nutritionnels.

Pour cela, les mécanismes d'élaboration et de préservation de la qualité d'un produit sont étudiés au sein des filières, qui sont prises en compte dans leur globalité. Les actions de recherche mises en place sont conduites en concertation avec les partenaires locaux et régionaux.

Le pôle KAPPA travaille selon trois axes de recherche :

- Gestion optimisée des systèmes d'élevage, qui a pour but d'améliorer la conduite intégrée de la ressource et de l'animal pour obtenir une production de qualité, par une évaluation de l'impact de conduites innovantes, et l'exercice d'une veille sanitaire au niveau régional.
- Production intégrée fruitière et horticole, qui apporte des solutions pour régulariser l'offre et la qualité des fruits sur les marchés, qui appuie la structuration des filières, et élabore des références techniques, économiques et organisationnelles pour obtenir une production de qualité.
- Élaboration de la qualité des produits alimentaires, par des procédés de transformation et des itinéraires techniques post-récolte innovants, en évaluant l'incidence des systèmes de production et de transformation sur la qualité sanitaire, nutritionnelle, organoleptique et technologique des produits animaux et végétaux.

Hiérarchie de la société

Conseil scientifique Président : Bertrand HERVIEU	**Conseil d'administration** Président : Gérard MATHERON	**Comité éthique** Président : Louis SCHWEITZER
Direction générale Gérard MATHERON		
Direction de la recherche et de la stratégie Patrick CARON		
Départements de recherche • Systèmes biologiques : Jean-Christophe GLASZMANN • Performances des systèmes de production tropicaux : Robert HABIB • Environnements et sociétés : Pierre FABRE		
Secrétariat général Patrick HERBIN		
Délégation aux évaluations Anne-Yvonne LE DAIN		
Délégation à la communication Anne HEBERT		
Délégation aux systèmes d'information Joël SOR		
Direction des relations européennes et internationales Jean-Luc KHALFAOUI		
Délégation à la valorisation Remy HUGON		
Délégation à l'information scientifique et technique Marie-Claude DEBOIN		
Délégation à l'enseignement supérieur et la formation Mireille MOURZELAS		
Directions régionales : • France : Guadeloupe, Guyane, Ile-de-France, Languedoc-Roussillon, Martinique, Nouvelle-Calédonie, Réunion. • Étranger : Afrique, Amérique latine, Asie.		

L'équipe

A la Réunion, le CIRAD compte 181 salariés permanents, dont 55 chercheurs, 120 techniciens et employés. A ces effectifs s'ajoutent une vingtaine d'ingénieurs volontaires civils à l'aide technique (VCAT) et autant de doctorants et de stagiaires. Si l'on additionne à ces chiffres les personnels non permanents et accueillis, cela fait plus de 250 personnes travaillant dans les locaux du CIRAD !
Chaque employé a son rôle et ses compétences, mais tous travaillent en étroite collaboration, et les projets de chaque sont finalement la contribution de tous, ce qui rend le travail très agréable, car les liens sont forts et l'accueil chaleureux !

Système d'information et environnement technique

Parmi les taches que réalise le CIRAD, nous retrouvons le développement d'équipements, d'outils et de logiciels afin de simplifier ou d'automatiser des processus. Très souvent ces produits forment une gamme d'outils, accompagnée d'expertise et de formation. Des responsables sont chargés de la promotion et de la distribution de ces outils, ainsi que de la formation des utilisateurs.

Le CIRAD conçoit et utilise de nombreux logiciels dédiés à l'agriculture, tels que des logiciels d'identification des espèces végétales, des pestes végétales, des logiciels biométriques, de statistique, de géo-référencement, de modélisation de la croissance des plantes, de base de données d'informations et de caractéristiques.
Parmi ceux que j'ai pu voir fonctionner pendant mes stages :

- Olympe: permet de créer une typologie de fonctionnement.
- R: l'analyse statistique.
- Gesmet : relevés météorologiques.
- Gen5...

Énoncé du problème

Pourquoi est-il important de bien comprendre l'intérêt de l'outil avant de le réaliser ?

Comment réécrire un logiciel dans un langage différent ?

Comment réaliser une application dans un environnement inconnu, dans un contexte nouveau ?

Qu'apporte l'utilisation de code orienté objet ?

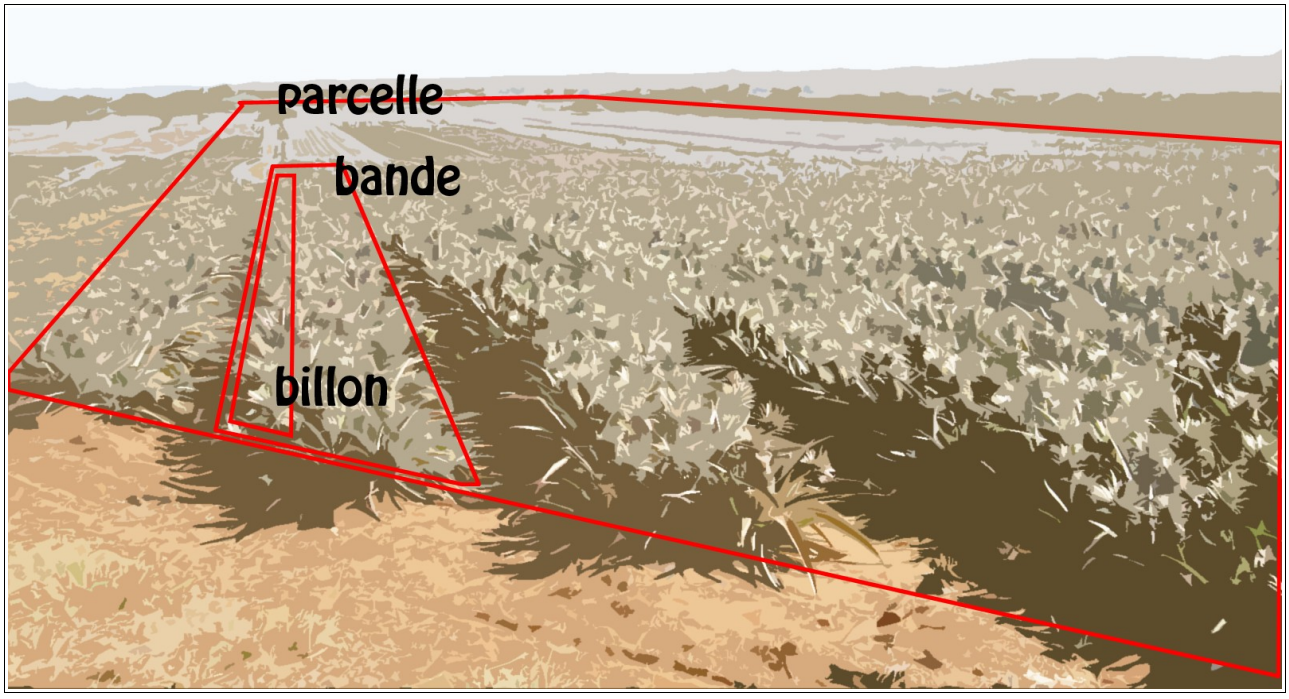
Comment traiter le problème avec une approche « objet » ?

Comment l'approche « objet » peut satisfaire les demandes en terme de modularité, de réutilisabilité du code, d'évolutivité, d'utilisation de librairies ?

Description du contexte

La gestion et le suivi d'une exploitation d'ananas a été modélisée de la manière suivante : voir ci-dessous le document « modélisation simplifiée de la gestion de culture d'ananas ». Mais ce modèle simplifié mérite encore quelques remarques.

Comme on peut voir sur le modèle existant, les parcelles et les bandes sont le centre de l'exploitation d'ananas. L'ensemble des renseignements saisis par les utilisateurs gravitent autour de ces deux piliers.
De par sa définition faite par IGN et l'Éducation Nationale sur l'information géographique, une parcelle cadastrale est une portion de territoire d'un seul tenant appartenant à un seul propriétaire et possédant une certaine individualité en raison de l'agencement donné à la propriété.
Chaque parcelle est composée de plusieurs bandes : on peut décrire les bandes comme des regroupements, et où l'espace entre ces bandes est suffisant pour circuler, en véhicule ou à pied.
Chaque bande est composée de plusieurs billons : un billon est une « ligne » où sont plantés de nombreux plants d'ananas. Les différents billons d'une bande sont très rapprochés.



Le TIF (Traitement d'induction florale) est un traitement qui permet la floraison artificielle. C'est à partir de la floraison du champ, rendue uniforme par le TIF qui permet de « forcer » la floraison,

qu'il est possible de prévoir la date de récolte et les quantités de production... Ces prévisions peuvent être faites grâce au suivi des conditions météorologiques et des traitements effectués sur le terrain.
Une autre utilité du logiciel est de déterminer un itinéraire technique à suivre : l'itinéraire technique est une liste des taches déjà faites et des taches à réaliser sur le terrain (par exemple un ajout d'engrais, un traitement contre les insectes, …).
Les rejets sont aussi traités par le logiciel, car ils donnent des précisions sur le déroulement de la prochaine plantation. Le rejet est la jeune pousse qui sera utilisée pour faire de nouveaux plants lors de la prochaine plantation, et qui apparaît après la coupe des fruits.
La météorologie fait partie intégrante du logiciel car elle est à la base de tous les calculs prévisionnels. Il y a donc une gestion détaillée des températures et des relevés pluviométriques.

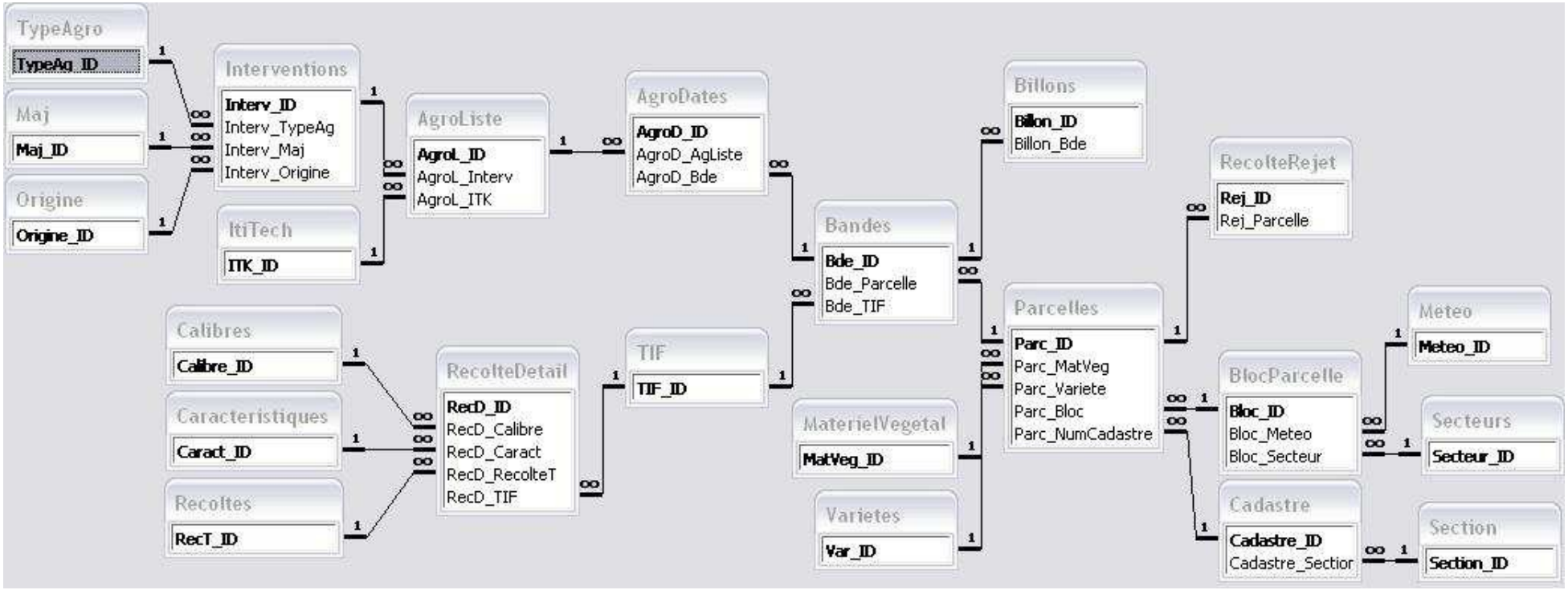
Après nous être familiarisés avec l''ancienne application Access, l'équipe composé de Mickaël FOLIO et de moi-même avons pu constater différents problèmes tels que :

- programmation procédurale
- la modification directe du contenu de la base de donnée par l'utilisateur
- présence de nombreuses parties de code redondantes
- le manque de flexibilité concernant l'importation météorologique
- la faible portabilité du logiciel, surtout concernant la facilité d'installation.

Ces critères sont les caractéristiques mêmes de ce qu'on appelle « programmation procédurale ». En effet le code est composé de routines, de procédures et de fonctions. Chaque élément de code réalise une opération spécifique, mais avec un réel manque d'interactions entre elles. Parmi les avantages trouvés, on a la possibilité d'appeler la même opération depuis plusieurs endroits, et cette technique permet de rendre la création et la lecture de l'algorithme assez faciles.
Le plus important constat que nous avons pu faire du logiciel existant, c'est qu'on remarque bien que le logiciel a été créé « au fil du temps », et selon les différents besoins qui sont apparus. C'est pour cette raison que des morceaux de codes deviennent redondants, et que le logiciel devienne de moins en moins « maintenable ».

Le but de notre projet consistait à refaire l'application en C#.NET, avec une approche en orienté-objet.

Modélisation simplifiée de la gestion de la culture d'ananas



Explication du choix du sujet

Ce sujet de stage semblait être un excellent défi pour l'équipe et le responsable de stage.

Le projet étant une transcription de l'application existante vers le C#.NET, il semble approprié de traiter l'ensemble du développement du projet basé sur ce langage de programmation. Le choix de la langue s'est fait en fonction de notre expérience sur de plus petits projets, ainsi que sur les cours SUPINFO et notre inscription au laboratoire .NET.

L'objectif du projet était de rendre l'application complète avec sa documentation utilisateur, ainsi que le nécessaire pour une installation sur les postes des utilisateurs, sans notre intervention.
Tout ce travail et ces recherches ont été utilisés pour rédiger ce mémoire.

Présentation des différentes méthodes pour traiter le problème

Les méthodes de travail permettant de résoudre le problème

Nous n'avions jamais travaillé sur d'aussi grands projets, et encore moins avec autant de données à stocker et traiter dans des bases de données. Il a donc fallu établir une solide méthode de travail pour réaliser le projet.
Pendant les quelques premières lignes de codes, peu liées au sujet, qui nous ont permis de tester les capacités et les limites du langage utilisé, nous avons tout mis en œuvre afin d'apprendre le maximum d'informations sur le sujet à traiter. Cette période, qui a été très longue, nous a vraiment formé sur la culture d'ananas et à toutes les notions qu'elle comporte. En effet, il est essentiel d'avoir ces connaissances sur le sujet : la maitrise du vocabulaire, des concepts, dans le but de faciliter la communication.

Le cahier des charges du projet a finalement été l'ensemble composé par l'application existante, la formation complète à toutes les fonctionnalités qu'elle comportait, ainsi que la compréhension de tout le processus sur le terrain comme sur les listes de taches.
Tout au long du projet les échanges avec l'employeur permettent de perfectionner encore plus ces directives, et de comprendre encore mieux le besoin qui doit être satisfait.

Après avoir bien compris le contexte du logiciel, il faut se pencher sur l'élaboration d'une base de données répondant au mieux au besoin, en prenant soin d'optimiser la manière dont seraient enregistrées les données, puisque leur grand nombre mettrait à rude épreuve les services de base de données.
Il existe plusieurs méthodes de modélisation de base de données (MERISE, UML, JACKSON,...) :

- Pendant cette phase il est important d'apprendre comment et pourquoi l'employeur a utilisé telle ou telle table, ainsi que chacun des champs qu'elle contient.
    - Les explications verbales sont claires mais des informations sont oubliées.
    - Les schémas informatisés sont complets mais parfois peu parlants.

    - Les notes des précédents développeurs, permettant de conforter les idées des différents partis.
- Il est important de toujours réfléchir en équipe sur la base de données, afin que l'avis soit partagé, cela permet d'éviter les modifications intempestives de la base.

Un fois la base de données terminée, avant de commencer le développement à proprement dit, il est nécessaire de découper le projet en plusieurs parties, et établir un plan de travail :
d'une part pour mieux répartir les taches entre les membres du groupe,
d'autre part pour mieux diriger le fil conducteur du projet, et faire en sorte de bien suivre l'idée du départ, idée qui a été bien réfléchie lors de l'étude de l'existant.

Pendant la phase de développement, il est intéressant d'écrire les fonctions sous forme algorithmique (ou pseudo-langage), afin qu'il soit plus facile de transcrite le logiciel, peu importe le langage cible.

A cette étape du développement, c'est le moment de coder, étape qui consiste à réécrire l'algorithme en C#.NET (dans notre cas), et à tester l'application pendant les différentes étapes de l'écriture (ce qui permet de suivre l'avancement des fonctionnalités ainsi que de veiller à garder la fonctionnalité de l'application).

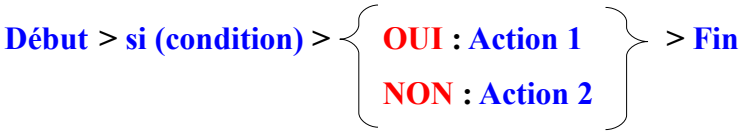
Comme cité plus haut, l'algorithme est une étape dans le développement du logiciel. Par définition, l'algorithme est une suite d'actions et de conditions qui définissent le comportement des éléments entre eux, et envers l'utilisateur.

Un algorithme peut être défini par trois scénarios :

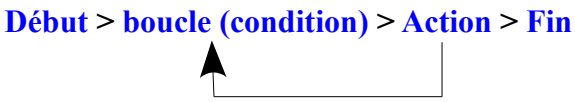
- Séquence : exécution d'instructions les unes après les autres.

Début > **Action 1** > **Action 2** > **Action 3** > **Action 4** > **Fin**

- Condition : Exécuter une opération plutôt qu'une autre, en fonction d'une condition définie.

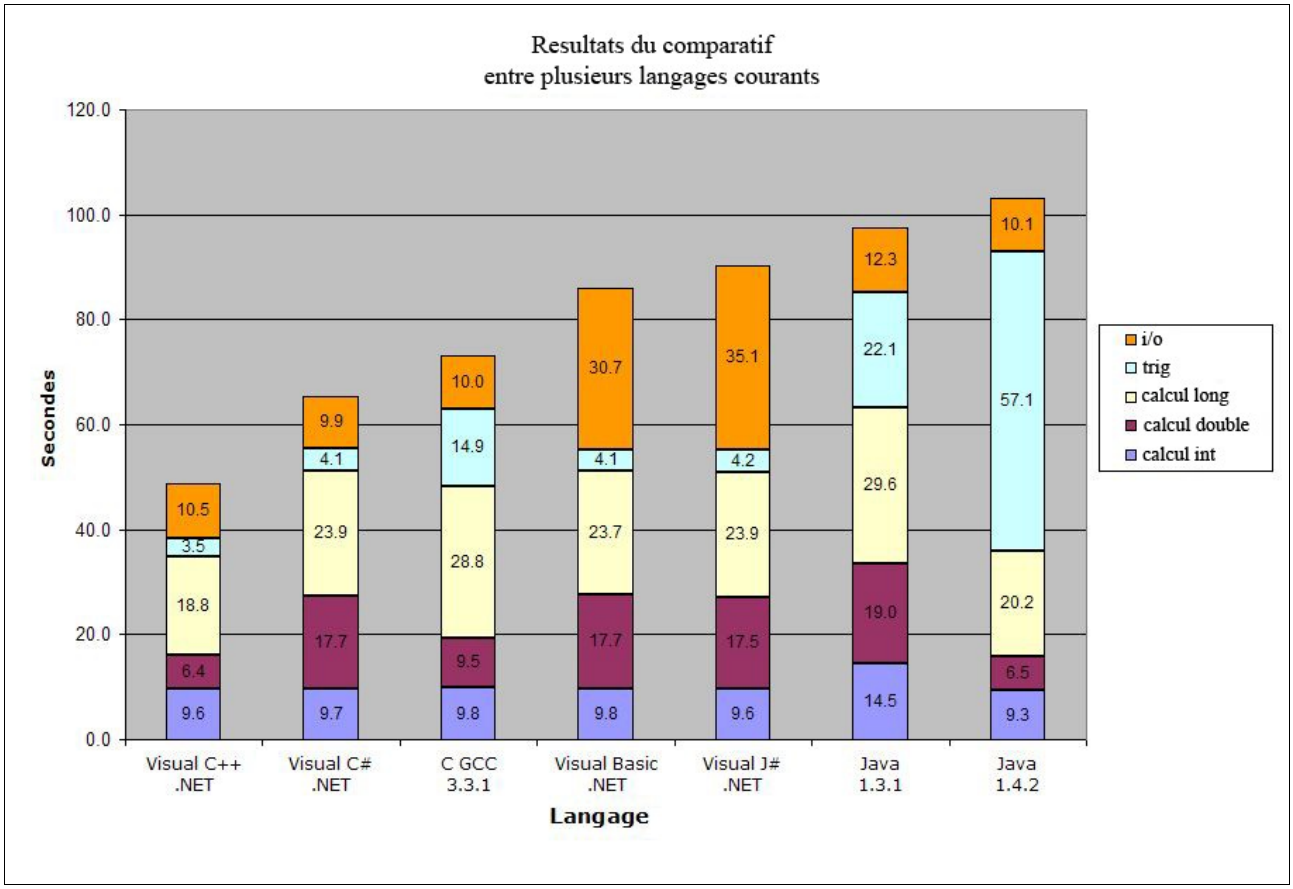


- Répétition : Effectuer la même opération un nombre déterminé ou indéterminé de fois.



Différences entre VB6 et C#.NET

Le site OSNews.com a fait une étude sur les performances d'exécution de différents codes avec les langages de programmations connus. On peut constater sur le graphique ci dessous que les langages .NET tirent leur épingle du jeu, et le C# est même très bien placé, derrière le C++ qui est le langage le plus proche de la machine.



Les langages .NET apportent de nombreuses améliorations par rapport aux versions précédentes telles que Visual Basic. En effet le .NET prend en charge les héritages entre les objets, les constructeurs, les surcharges de méthodes, la gestion avancée des exceptions, des threads, la présence de code managé, ainsi que de nombreuses autres fonctionnalités.

Au niveau de l'environnement de développement, la plateforme Visual Studio est de loin la plus complète, et offre une panoplie d'outils tels que le concepteur visuel, XML, l'explorateur de serveur, des outils de création pour le déploiement, de métrologie du code, de mesure de performance, de test unitaire, et plus récemment du développement collaboratif.

Le Framework .NET offre ainsi de nombreux outils permettant la rédaction plus rapide de code, tout en écrivant moins de lignes. En effet de nombreuses méthodes intégrées au Framework permettent un traitement rapide des données, avec un code optimisé : par exemple les méthodes de tri, de recherche et de sélection, de conversion... Ce sont des inconvénients du C++ que le C# résout grâce au Framework. De nombreuses modifications ont été apportées sur les différents contrôles utilisateur, de façon à généraliser les propriétés, tout en gardant la puissance d'une union des propriétés et des méthodes, au sein d'un « objet ».

Une des contraintes du C# est qu'il est nécessaire de faire appel à l'objet et à la propriété voulue pour y avoir accès, alors qu'en VB6 il était possible d'avoir la propriété par défaut en appelant juste le nom du contrôle. Mais Le code devient en C# beaucoup plus strict et donc plus lisible.
Le C# est finalement du code C++ (C orienté-objet), avec tous les avantages du C, et avec toutes les facilités et les aides qu'a apporté le Framework .NET.

Le C#.NET apporte aussi ces changements sur le VB6 :

- l'apparition de ADO.NET et Entity Framework pour l'accès aux données
- GDI+ pour le traitement graphique (GDI simple pour VB6)
- Certaines notions en terme de graphique ont disparu à cause de la nouvelle approche objet
- Le presse papier supporte de nombreux formats
- Le travail multithread est possible, avec le contrôle, la communication et l'interaction entre les threads
- Le polymorphisme, les surcharges, l'héritage sont présents et applicables aux objets

- Les méthodes doivent obligatoirement être suivies de parenthèses (plus de confusion avec les variables)
- Les méthodes peuvent renvoyer des variables de tout type avec le mot clé Return.
- Les contrôles ne peuvent plus être groupés mais sont contenus dans des GroupBox ou Panels
- Certains contrôles ont bien été dissociés pour une meilleur utilisation.
- Contraintes positives du C : on ne peut plus changer de type de variable en cours de route.
- Chaque variable doit être explicitement déclarée pour être utilisable.
- Toute variable de base est de type « object » par défaut (« variant » en VB6).
- Apparition de collections, listes, utilisées aussi dans les contrôles comme Listbox, Listview, offrant une meilleure gestion de la mémoire, notamment lorsque le nombre d'éléments change.
- Les nombres Entiers (Int) sont désormais enregistrés sur 32 bits, et les Longs sur 64 bits.
- L'utilisation de booléens ne se fait plus qu'avec les mots clés True et False (chiffres 0 et 1 autorisés en VB6).
- La déclaration de plusieurs variables pour un seul type est désormais possible :
  - en VB6, « Dim a, b, c as Integer » déclare a et b en variant, et c comme entier.
  - En C#.NET « int a, b, c; » déclare a, b et c comme des entiers.
- Un type string n'a désormais plus de taille fixe en mémoire.
- La portée d'utilisation d'une variable peut désormais être restreinte à un bloc.

```
{
    int a;
    {
        int b;
        // a est accessible ici
        // b est accessible ici
    }
    // a est accessible ici
    // b n'est pas accessible ici
}
```

De manière générale, on a tendance à penser que la présence du Framework derrière l'application

entraine un ralentissement de l'exécution du code. Mais le raisonnement est différent : Le fait que le code soit aussi compilé, qu'il soit très optimisé, notamment avec la présence des objets, le fait aussi qu'on utilise des outils plus avancés dans l'environnement de travail... tout cela entraine une grande avancée dans la vitesse de rédaction, et ce code ainsi optimisé exécute finalement plus rapidement les opérations, tout en utilisant de manière plus efficace la mémoire.

Pour conclure, on peut dire que la réécriture du code en orienté-objet doit être considérée de manière large sur l'application, et il faut optimiser au maximum le code en utilisant les fonctions prédéfinies offertes par le Framework .NET. Cela en fait donc une application légère, complète, et avec une bonne gestion de la mémoire et des erreurs.

L'approche Objet

L'objectif du projet a été de repenser l'application entièrement, pour que le fonctionnement global exigé soit entièrement basé sur l'approche orienté-objet, afin que l'utilisation du langage C#.NET soit optimisée (il faut tirer le meilleur parti possible des possibilités offertes).
Programmer en orienté-objet signifie utiliser un maximum de classes pour gérer les informations en mémoire, pendant le développement. Il n'est pas évident pour un développeur n'ayant jamais développé en objet de le faire, il est donc important de bien se documenter sur Internet et de s'entrainer sur des sujets simples, ou de petits projets. Les cours SUPINFO notamment en XNA (jeux vidéos en C#.NET) m'ont permis de vraiment comprendre l'intérêt et l'utilisation des objets, c'était finalement le moyen le plus concret de « voir » un objet, puisque la plupart des « objets code » étaient finalement visuellement des « objets du jeu ». Ainsi quand on codait une voiture dans les sources du jeu, on voyait bien une voiture à l'écran, avec les propriétés et les actions qu'on avait nous-mêmes écrites dans le code. C'est à partir d'une vision concrète et réelle des objets qu'il devient simple et évident de coder des objets dans tout logiciel, même aux endroits où les objets n'ont plus de forme réelle. Le travail abstrait devient simple.

Par définition, un objet est une instance d'une classe, mise en mémoire. Cela inclut :

- un nom
- un comportement grâce à des méthodes qu'il peut exécuter
- un état, grâce à des propriétés qu'on peut lire et modifier

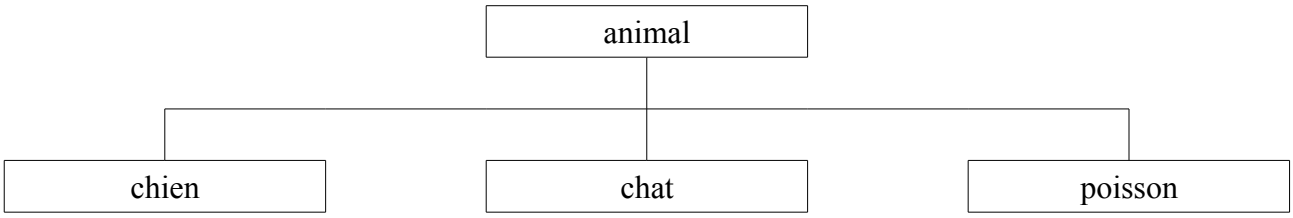
Les avantages de l'utilisation des objets sont :

- la réutilisabilité du code (un objet peut être utilisé n'importe où dans le code, du moment qu'il est accessible)
- la réduction de la complexité du code (passer un objet en paramètre ou en résultat d'une méthode est bien plus simple et plus léger que de passer l'ensemble des propriétés de cet objet)
- encapsulation des variables : permet de « surveiller » l'accès aux variables, et éventuellement de les protéger, ou d'agir en cas de modification. Exemple : Il n'est pas nécessaire au conducteur de savoir tout le mécanisme du moteur pour démarrer un véhicule, il lui suffit de tourner la clé.

Une classe est utilisée pour modéliser des objets réels. Ce concept correspond au regroupement de toutes les informations qui caractérisent cet objet. Prenons un exemple simple :

- objet : un chien
- propriétés : nombre de pattes, couleur, grandeur, poids...
- Comportement : marche, court, aboie, mange...

La programmation orienté-objet permet aussi la création de classes abstraites. Ces classes sont utilisées pour regrouper des caractéristiques communes à plusieurs objets. Si on continue avec l'exemple ci-dessus on obtient :



On peut voir sur cet exemple que les objets chien, chat et poisson héritent des caractéristiques de la classe abstraite animal. Une classe abstraite ne peut pas être initialisée (on ne peut pas en faire un objet), elle peut simplement servir de « modèle » pour la création de classes dérivées.

Maintenant que l'approche orienté-objet a bien été définie, il faut savoir comment définir les objet qui seront utilisés dans notre projet. On doit :

- identifier l'ensemble des objets
- déterminer leurs comportements
- déterminer les relations qu'auront les objets entre eux

Quand toute cette phase d'identification est terminée, il est possible d'établir réellement la base de données, et de mettre en forme les différents algorithmes des opérations.

Explication des méthodes retenues pour résoudre le problème

Introduction au .NET

L'objectif retenu de mes différents stages au CIRAD (y compris le stage de fin d'études) était de réécrire l'application qui avait initialement été programmée en Visual Basic sur Access, en une application autonome sur la plateforme .NET. Le langage choisi de cette plateforme a été le C#, qui offre les meilleurs performances, et il avait été enseigné et beaucoup utilisé à SUPINFO. Le C#.NET est un langage créé par Microsoft, développé sur la plateforme de développement Visual Studio, et permet le développement d'applications à destination des systèmes Microsoft Windows.

Visual Studio est un outil de travail visuel qui permet de créer des interfaces visuelles (GUI : Graphic User Interface) sans écrire de lignes de codes, en utilisant un éditeur simple (les boutons, images, zones de texte et menus se créent, se déplacent et se personnalisent simplement avec la souris).

L'avantage d'un tel langage et de pouvoir, pour chaque évènement (action) possible sur chacun des contrôles « visuellement » créés, attribuer une portion de code (exemple, cliquer sur un bouton, appuyer sur une touche, survoler une zone avec la souris...).

Pour écrire concrètement un logiciel, il faut donc passer par deux étapes :

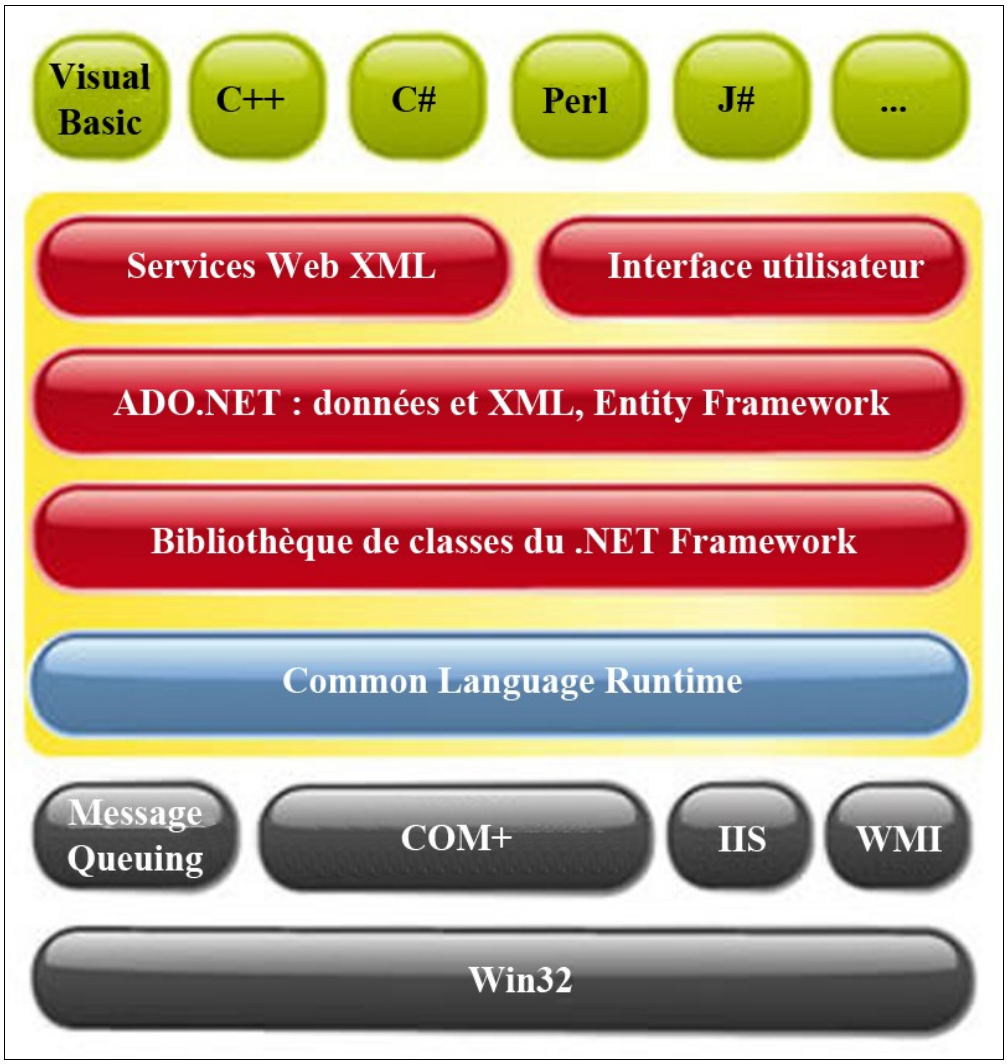
- mettre en place l'interface, donc tous les éléments visuels qui la composent.
- Écrire le code à exécuter pour chacune des actions désirées de ces éléments.

Le C#.NET est une mise à jour du C et du C++, apportant donc les avantages du code orienté-objet du C++, ainsi que toutes les classes prédéfinies et autres méthodes du .NET Framework de Microsoft. Il est donc possible en C# de coder une application mêlant l'interface graphique simplement dessinée, le code procédural des événements, et la gestion « objet » des éléments en mémoire et leur transition entre les différents traitements.

Le .NET Framework utilise ce que l'on appelle le CLR (Common Language Runtime), un environnement managé qui fournit des services de sécurité, une gestion de la mémoire (comme par exemple le Garbage Collector, gestion automatique de la mise en mémoire et de la libération de mémoire par les variables et objets), ainsi qu'un accès aux données via l'ADO.NET et l'Entity Framework. Ce sont les principales technologies utilisées pour réaliser ce projet pendant les stages.

Les classes fournies par le .NET Framework forment une couche qui lie le système d'exploitation (Windows) avec l'application C#. L'inconvénient majeur du Framework est sa lenteur (puisqu'il contient des milliers de classes déjà codées), mais il offre par contre de nombreux avantages :

- Il n'est installé qu'une seule fois.
- Il peut être utilisé par tous les différents langages de la plateforme Visual Studio
- Il fournit une indénombrable quantité d'outils et de traitements au développeur.

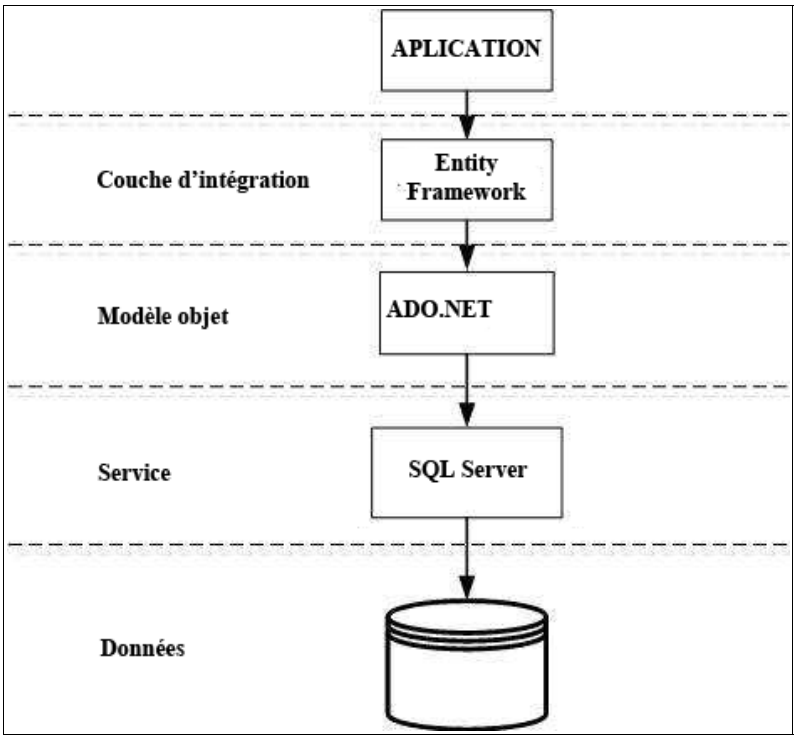


ADO.NET est une méthode d'accès aux sources de données.

- ADO (ActiveX Object Database) est la couche d'accès à la base de données.
- L'extension .NET signifie que la couche est gérée par le CLR du Framework.
- ADO.NET s'utilise avec un seul et même langage, quelque soit le type de la base de données, ou son contenu.
- L'extension Entity Framework permet un accès sans utiliser de langage spécifique aux bases de données (SQL).

Pour accéder à la base de données, il est nécessaire de charger en mémoire les pilotes spécifiques correspondant au type de base de données utilisée. Dans le projet du CIRAD, la base de données utilisée est sur la plateforme SQL Server Express.

Pour accéder à la base de données, il faut établir une connexion avec elle. Chaque connexion (une connexion par source de données) doit posséder une « ConnectionString » qui définit les paramètres à utiliser pour l'accès à la base (chemin d'accès, identifiant, mot de passe, etc.). A chaque source de données sa chaine de connexion. Pour savoir comment rédiger cette chaine (que l'on peut retrouver dans le fichier XML des paramètres de l'application nommé *App.Config*), de nombreux documents existent sur Internet. De plus, Visual Studio intègre maintenant des assistants pour rechercher et paramétrer automatiquement l'accès à la base de données.



Pour accéder à la base de données il faut respecter un algorithme précis :

1. Ouvrir l'accès à la base de données.
2. Lire des éléments dans la base de données.
3. Manipuler les données de la base de données : insertion, mise à jour, suppression...
4. Fermer l'accès à la base de données.

Toutes ces manipulations se font dans les lignes du code de l'application : le code C#.

Introduction au C#.NET

Même lorsque l'application est orienté-objet, il est nécessaire de diviser l'ensemble des actions en plusieurs parties. Ces différentes parties forment des méthodes, chacune ayant un rôle dans une suite d'événements, ou un déclenchement par un événement utilisateur (clavier, souris, …).

En C#, les méthodes sont écrites comme en C, à savoir :

```
private void HelloWorld()
{
        textBox1.Text = "Bonjour tout le monde !";
}
```

Dans ce cas la méthode ne renvoie pas de résultat, et ne prend aucun paramètre. Ci dessous la même méthode avec des éléments supplémentaires :

```
private bool HelloWorld(string nomUtilisateur)
{
        textBox1.Text = "Bonjour " + nomUtilisateur + " !";
        return True;
}
```

Cette méthode prend comme paramètres d'entrée une chaine contenant le nom de l'utilisateur, et renvoie VRAI à la fin du traitement.
Les blocs sont délimités avec les ouvertures { et les fermetures }.

Le passage de paramètres à une méthode peut être fait de trois façons :

- « byval » : la variable est copiée et la copie est utilisée dans la méthode. Si on modifie la variable dans la méthode, elle ne change pas dans la méthode appelante.
  Utilisation : private void HelloWorld(string nomUtilisateur);
- « byref » : un pointeur vers la variable est passée à la méthode, si dans cette dernière la variable est modifiée, le changement est donc aussi effectué dans la méthode appelante.
  Utilisation : private void HelloWorld(ref string nomUtilisateur);
- Variable de sortie : un pointeur vers la variable est fourni à la méthode, pour écriture uniquement. On peut ainsi dans la méthode attribuer une valeur à cette variable, que la méthode appelante pourra réutiliser par la suite. C'est utile dans les cas où plusieurs informations doivent être renvoyées à la méthode appelante. Mais il est vivement déconseillé d'utiliser cette technique, qui n'est plus d'actualité avec les avantages qu'offrent la programmation orienté-objet.
  Utilisation : private void HelloWorld(out string nomUtilisateur);

Tout type de données peut être passé en paramètres à une méthode : des images, des tableaux d'objets, des collections, … De même tous les types sont autorisés pour les retours de méthodes.

Les conditions sont écrites exactement comme en C :

```
if (condition)
{
        ActionSiOui();
}
else
{
        ActionSiNon();
}
```

On peut imbriquer un nombre indéfini de conditions, en respectant bien de préférence l'indentation du code, et la séparation en blocs avec les signes { et }. La lecture reste ainsi claire et ordonnée.

Concernant les boucles, plusieurs types existent.

- Une boucle avec un nombre incrémenté (pour les cas où le nombre de boucles est connu)

```
for (int i = 0; i < 100; i++)
{
        ActionAFaire100Fois();
}
```

- Une boucle sur une liste d'objets connus, chaque boucle traitant un objet de la liste.A chaque tour de la boucle, l'objet suivant de la liste des Variétés est mis dans la variable "v", et le contenu du bloc est executé :

```
foreach(Variete v in monAcces.Variete.ToList())
{
        string nom = v.nom;
}
```

- Pour exécuter une action un nombre indéterminé de fois selon une condition (il faut noter que cette boucle exécute l'opération 1 fois avant de vérifier si la condition est remplie) :

```
do
{
        ActionAFaireTantQueConditionVraie();
} while (condition);
```

- La même boucle qui vérifie la condition avant de faire l'opération (si la condition est fausse, l'action ne sera pas effectuée) :

```
while (condition)
{
        ActionAFaireTantQueConditionVraie();
}
```

Ces quelques notions permettent déjà de réaliser bon nombre d'opérations en C#. Ces mêmes lignes donnent aussi des notions solides sur la syntaxe du code C et C++.

SQL Server

Afin de traiter les données utilisateur avec efficacité, il convient de stocker ces données de manière judicieuse. La base de données est dans notre cas indispensable, puisque les données sont abondantes. Comme précisé précédemment, les services de bases de données retenus pour le projet du CIRAD sont SQL Server de Microsoft. L'outil s'intègre parfaitement au système, son installation est simple, et son poids raisonnable (une centaine de mégaoctets).

Le souhait de pouvoir copier la base de données comme un simple fichier, dans le dossier de l'application, a entrainé le choix d'utiliser une version spéciale : SQL Server Compact 3,5. En effet la version Compact permet le « montage » d'un fichier base de données lors du démarrage du logiciel, et la libération de celui-ci lors de la fermeture. Il est ainsi plus simple de faire des copies, sauvegardes et restaurations des bases de données des utilisateurs (surtout pendant la phase de recherche d'anomalies de code). Toute l'application, et l'ensemble de ses données, est ainsi contenue dans le dossier d'installation du logiciel.

L'outil Microsoft SQL Server Management Studio disponible aussi gratuitement que Microsoft SQL Server Express, permet l'accès aux bases de données montées et hébergées, en local comme à distance, pour la consultation comme pour la modification de leur contenu.

Entity Framework

Entity Framework est un Framework du .NET Framework.
C'est une innovation apparue depuis le Framework 3.5 SP1 dont voici la définition (source wiki) :

Entity Framework est un mapping objet-relationnel (en anglais « object-relational mapping » ou ORM), c'est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé. On pourrait le désigner par « correspondance entre monde objet et monde relationnel ».

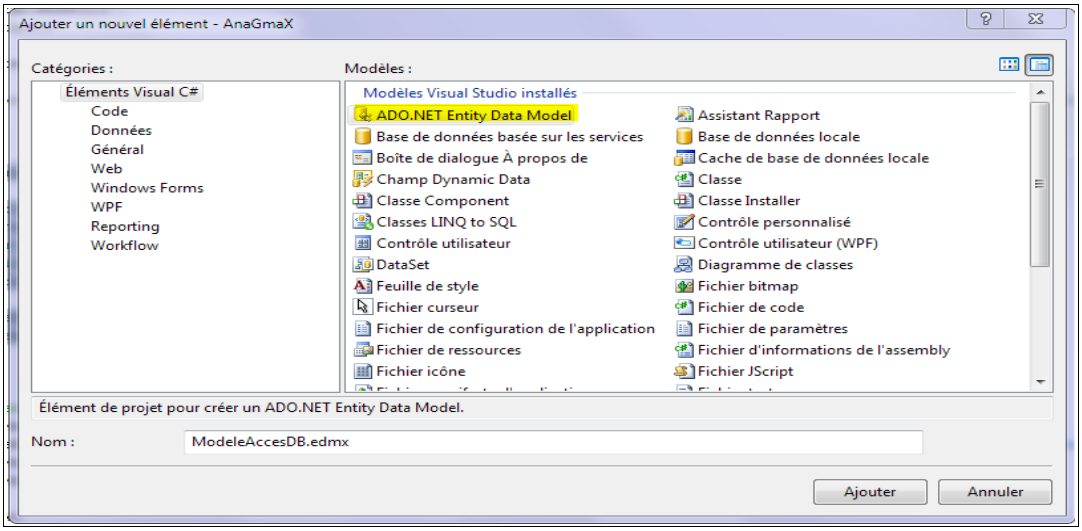
Pour résumer, c'est une couche du Framework qui permet à l'utilisateur de travailler sur une base de données relationnelle en tapant du code objet. Ce Framework crée l'ensemble des classes, y compris toutes les commandes pour lire, écrire, modifier et supprimer des éléments dans la base de données.
L'utilisation des éléments de la base de données devient ainsi beaucoup plus simple, et évite de nombreuses vérifications usuelles lors de la lecture, modification et suppression d'éléments dans la

base de données.

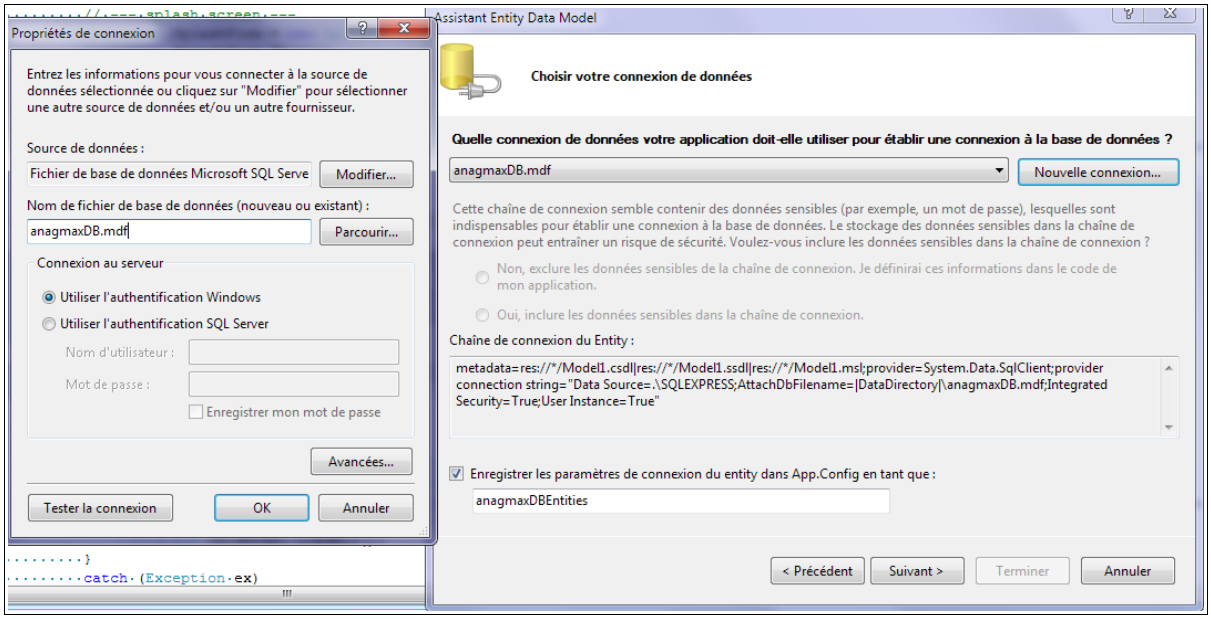
Cette technologie a été une véritable innovation puisqu'elle permet de gagner des heures de développement (et de plus exponentiellement proportionnelles au nombre de tables des bases de données) et permet de pousser très loin les limites d'utilisation de la base, puisqu'il n'y avait plus nécessité de saisir des requêtes manuellement.

L'accès aux données est créé avec l'assistant intégré de Visual Studio :

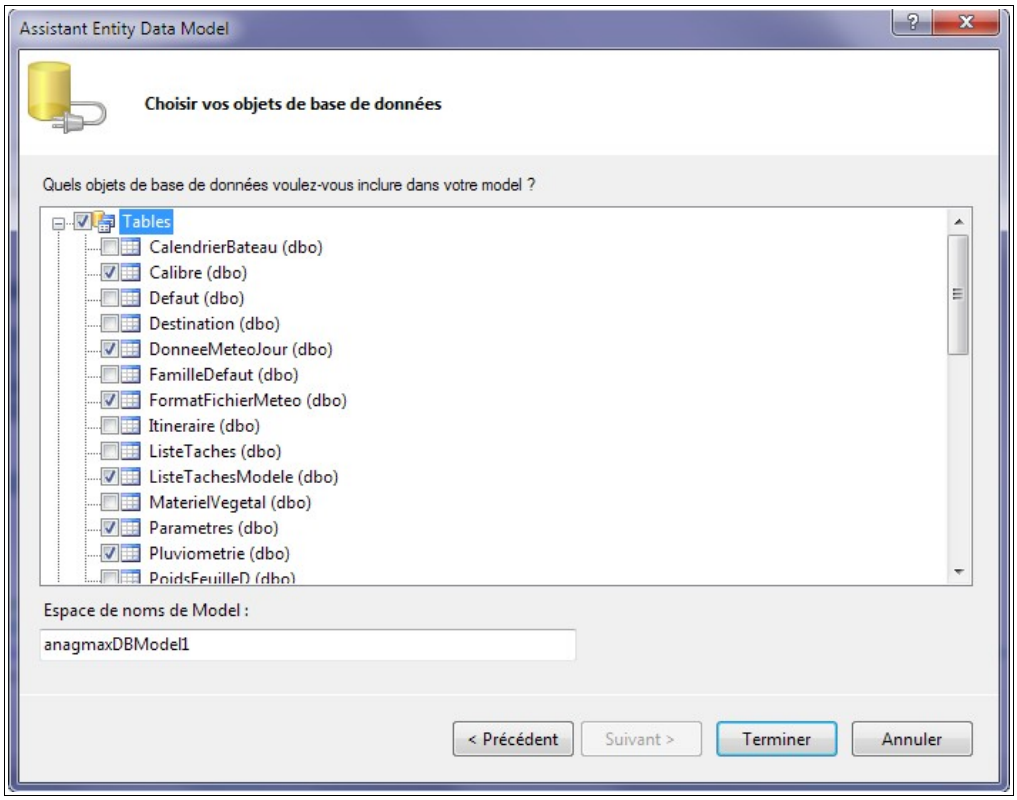
1. Création d'un ADO.NET Entity Data Model.



2. Choix de la base de données locale ou distante à partir de laquelle créer le modèle.
    1. Création de la chaine de connexion : Choix du fichier de base de données Microsoft SQL Server (SqlClient). Il faut saisir un nouveau nom, ou repérer un fichier existant.
    2. Saisie d'un nom de paramètres du modèle Entity. Exemple « anagmaxDBEntities ».

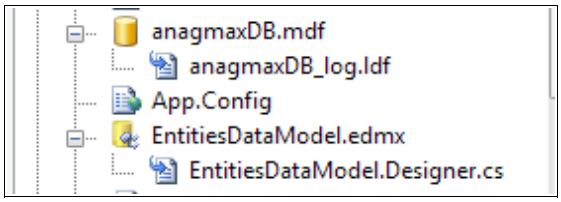


3. Choix des tables à inclure dans le modèle.



4. Le modèle est créé, et un diagramme complet est affiché. Depuis ce diagramme on peut modifier les types si un changement est à faire (attention à la compatibilité avec la base de données lors des changements !). On peut aussi ajouter ou supprimer des relations entre les tables. Pour finir il est possible de définir des relations d'héritage, chose que la base de données ne peut pas gérer d'elle même, mais qu'il est possible de faire avec Visual Studio si on a prévu tous les champs dans la base.

Liste des fichiers créés (nom selon le contexte) lors de la création du modèle :



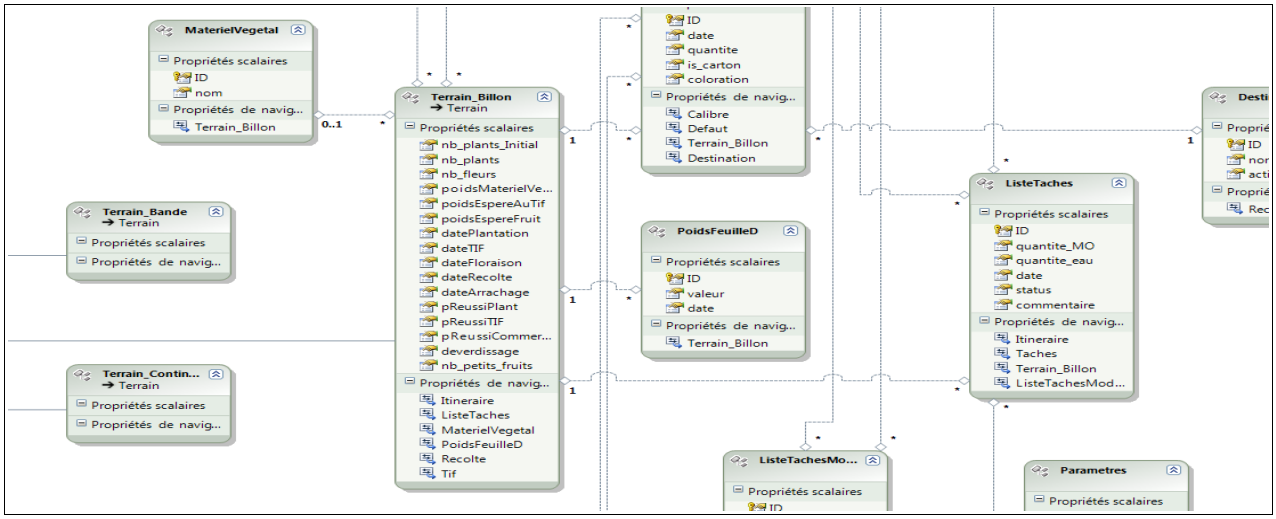
La base de données (anagmaxDB.mdf) et son fichier de log (anagmaxDB_log.ldf) sont créés si non existants, sinon le fichier peut être récupéré dans le dossier Data du serveur SQL.

Exemple : *C:\Program Files (x86)\Microsoft SQL Server\MSSQL.1\MSSQL\Data\anagmaxDB.mdf*

Le fichier App.Config contient donc la chaine de connexion expliquée précédemment :

```
<?xml version="1.0"?>
<configuration>
  <connectionStrings>
    <add name="anagmaxDBEntities" connectionString="metadata=res://*/EntitiesDataModel.csdl|
res://*/EntitiesDataModel.ssdl|res://*/EntitiesDataModel.msl;provider=System.Data.SqlClient;provider connection
string=&quot;Data Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\anagmaxDB.mdf;Integrated
Security=True;Connect Timeout=30;User Instance=True;MultipleActiveResultSets=True&quot;"
providerName="System.Data.EntityClient"/>
  </connectionStrings>
  <startup><supportedRuntime version="v2.0.50727"/></startup>
</configuration>
```

Le fichier EntitiesDataModel.edmx et son fichier Designer.cs associé contiennent tout le code qui a été généré pour faire le lien entre la base et les nouveaux objets disponibles dans le code. Le Designer permet l'affichage du diagramme, dont voici un extrait ci-dessous :



Il faut au final retenir que pour avoir toute la puissance de Entity Framework dans une application, il est nécessaire de bien penser sa base de données, de la créer entièrement avec les outils adéquats (SQL Server Management Studio dans notre cas), et le reste se fait tout seul! Concernant le code, puisque toute la partie base de donnée est traitée automatiquement, il ne reste qu'à utiliser les données...

Ouverture de session

Pour ouvrir la connexion il faut créer un objet à partir de la classe nouvellement créée :

```
DBEntities monAcces = new DBEntities();
```

Il est désormais possible d'accéder à tout le contenu de la base de données, rien de plus compliqué !

Lecture des données

Pour lire la liste des matériels végétaux on peut donc simplement saisir :

```
anagmaxDBEntities monAcces = new anagmaxDBEntities();
List<MaterielVegetal> listeMateriels = monAcces.MaterielVegetal.ToList();
```

Pour récupérer un élément spécifique (aussi bien qu'une liste), on peut réaliser l'opération de deux façons :

- le Linq To Entity :

```
Variete variete = (from v in monAcces.Variete.ToList()
                   where v.nom == "variété de démonstration"
                   select v).First();
```

- les expressions lambda :

```
Variete variete = monAcces.Variete.Where(v => v.nom == "variété de démonstration").First();
```

Le Linq-To-Entity permet une belle structure du code et une bonne compréhension, notamment pour la relecture. Son inconvénient est qu'il faut parfois avoir eu de bonnes notions de SQL pour pouvoir rédiger les requêtes. C'est la méthode la plus pratique lorsque de nombreux paramètres doivent être spécifiés.
Les expressions lambda permettent de faire au plus court pour arriver à un résultat. C'est la méthode conseillée pour récupérer une liste de sous éléments ou un élément, lorsque la demande ne nécessite que peu ou aucun argument particulier.

Création d'éléments

Pour créer un nouvel élément dans la base, quelque lignes suffisent également :

```
anagmaxDBEntities monAcces = new anagmaxDBEntities();
Variete nouvelleVariete = new Variete();
nouvelleVariete.densiteParDefaut = 30;
nouvelleVariete.nom = "variété de démonstration";
nouvelleVariete.poidsEspereRecolte = 1200;
nouvelleVariete.zeroVegetatif = 12;

monAcces.AddToVariete(nouvelleVariete);
monAcces.SaveChanges();
```

La deuxième ligne crée comme toute déclaration de variable, un espace en mémoire pour l'objet de type Variété. Les quatre lignes suivantes attribuent des valeurs aux propriétés de l'objet nouvelleVariete. L'avant-dernière ligne permet explicitement l'ajout de l'objet en mémoire dans la base de données. Il faut savoir que cette opération est faite en tache de fond même sans utiliser cette ligne de code. La dernière ligne de code permet la sauvegarde de toutes les modifications faites dans la base de données. C'est une opération à ne jamais oublier !
Il est donc au final très rapide de créer un objet dans la base : une ligne pour l'instancier, une ligne pour sauvegarder, tout le reste est implicite ou optionnel.

Modifier un élément

Pour modifier un élément dans la base, on le récupère comme vu précédemment dans la lecture des données, on modifie les champs désirés, et on sauvegarde. Avec ces quelques lignes le travail est effectué :

```
Variete variete = monAcces.Variete.Where(v => v.nom == "variété de démonstration").First();
variete.nom = "variété de démonstration 2";
monAcces.SaveChanges();
```

La modification est mémorisée, et cela de manière permanente. Il est par contre vivement conseillé

de ne lancer la sauvegarde qu'à la fin de toutes les modifications, car la sauvegarde est l'opération la plus longue. De même lors de la modification ou de l'insertion d'un grand nombre de données, il est important de sauvegarder de manière régulière plutôt qu'à la fin, car la charge de travail peut parfois prendre tellement de temps que l'application semblera figée. Par exemple une sauvegarde tous les 500 ou 1000 éléments permettra l'entretien d'une barre de progression, et ainsi rendra l'application plus conviviale et rassurante sur les grosses taches.

Suppression d'éléments

La suppression d'éléments est toute aussi simple. Il faut d'abord récupérer l'objet à supprimer (lecture dans la base) puis demander au modèle de données de le supprimer. Ces quelques lignes suffisent pour exécuter l'opération :

```
Variete variete = monAcces.Variete.Where(v => v.nom == "variété de démonstration").First();
monAcces.DeleteObject(variete);
monAcces.SaveChanges();
```

Pour supprimer un lot d'informations, comme par exemple le contenu d'une table, on peut combiner les différentes approches vues précédemment :

```
monAcces.Variete.ToList().ForEach(v => monAcces.DeleteObject(v));
```

Cette ligne, à première vue très complexe, récupère en fait la liste des variétés. Le ToList est nécessaire pour créer une liste « figée » d'objets, si on ne l'utilise pas, à la suppression du premier élément la liste mémoire se mettrait à jour et le ForEach renverrait une erreur (car il ne travaille pas sur des listes dynamiques). Le ForEach va exécuter le code qui suit pour chaque élément de la liste. L'instruction de suppression est la même que plus haut. Ce code peut être remplacé par un code plus simple (sans expressions lambda), et ayant exactement le même résultat :

```
foreach (Variete v in monAcces.Variete.ToList())
{
        monAcces.DeleteObject(v);
}
```

Ce qui nous montre une fois de plus l'intérêt des expressions lambda.
L'essentiel concernant la création et l'utilisation des modèles de données Entity Framework a été vu, ces bases permettent de développer en très peu de temps des applications utilisant de grosses bases de données. Il n'est désormais plus nécessaire d'en connaître davantage pour aux données de la base, il faut juste écrire le code pour traiter les données lues, pour les modifier, les supprimer, et les sauvegarder.

Liaison DataSource

La dernière chose importante à savoir sur l'utilisation des données, est qu'on peut facilement remplir une liste visuelle (ListBox) à partir d'une liste d'objets ou de la base de données directement. C'est l'utilisation des DataSources. Avec ces quelques lignes :

```
listBox1.DataSource = monAcces.Variete.ToList();
listBox1.DisplayMember = "nom";
```

Le ListBox va se remplir à partir des objets de la table Variété, et le champ “nom” des objets sera utilisé pour le texte affiché sur chaque ligne. Ainsi la liste se remplit toute seule, et lorsque l'on souhaite récupérer un élément dans la liste, par exemple celui sélectionné, on peut utiliser du code comme celui-ci :

```
Variete selection = listBox1.SelectedItem as Variete;
```

L'objet mémoire “selection” sera donc l'objet qui avait été sélectionné dans la liste.
On utilise le “as” pour forcer l'identification du contenu de la sélection, puisque pour pouvoir contenir n'importe quel objet, la liste ne contiendra en fait que des objets sous leur forme primitive “object”.
Il faut noter que le DataSource est présent sur de nombreux controles, tels que Label, TextBox, ComboBox, … Et permet donc un fort gain de temps sur l'affichage des données pour l'utilisateur.

Calculs sur les données de la base

Une autre utilisation utile des données est : avec les calculs. En effet il n'est pas nécessaire avec le Framework .NET, de faire des boucles pour des calculs simples sur les données. On peut récupérer un champ particulier d'une sélection d'objets pour faire une liste de ce champ.
Au lieu de créer une liste de 1000 variétés pour calculer une valeur sur ces 1000, on peut créer une liste de 1000 valeurs qu'on va traiter.
L'opération se fait ainsi :

```
List<int> listeValeurs = monAcces.Variete.Select(v => v.zeroVegetatif);
```

On aura donc une liste de int qui contient la valeur du zéro végétatif de chaque variété.
Pour récupérer la moyenne de cette liste, on peut donc simplement faire :

```
int moyenne = listeValeurs.Average();
```

On peut effectuer le même résultat sur une seule ligne de code :

```
int moyenne = monAcces.Variete.Select(v => v.zeroVegetatif).Average();
```

Cette opération est simple, rapide, et occupe le strict minimum en mémoire, puisque c'est le moteur de base de données qui va réaliser l'opération, et non pas l'application. Le procédé complexe a été mis au point sur les Framework récents, et permet un calcul des données en différé, selon le moment d'accès à la variable. Les expressions lambda et le LINQ permettent l'utilisation différée, en étroite liaison avec le Entity Framework.

Pour faire un calcul de somme, la procédure est aussi simple :

```
int somme = monAcces.Variete.Select(v => v.zeroVegetatif).Sum();
```

Pour connaître le nombre de valeurs de la liste (dans la base) :

```
int nbVals = monAcces.Variete.Select(v => v.zeroVegetatif).Count(); // c'est la demande de calcul dans la base
```

Pour connaître le nombre de valeurs de la liste (en mémoire) :

```
int nbVals = uneListe.Count; // c'est une propriété de la liste
```

Pour récupérer la plus petite valeur :

```
int moyenne = monAcces.Variete.Select(v => v.zeroVegetatif).Min();
```

Pour récupérer la plus grande valeur :

```
int moyenne = monAcces.Variete.Select(v => v.zeroVegetatif).Max();
```

Flux de données : les fichiers

Pouvoir lire et stocker des informations dans la base de données est une bonne chose, mais parfois l'utilisateur ne souhaite pas forcément saisir ces données une à une. Il faut donc qu'il puisse importer des données d'un autre logiciel, et de manière simple à travers un format lisible et universel. Le format CSV a ces propriétés. Le CSV (Comma-Separated Values) est un fichier texte (simple) donc les différentes propriétés d'une information sont séparées sur la même ligne par point-virgule. L'évolution de l'informatique a fait que les CSV puissent non plus être restreints à leur description native, mais que leur utilisation soit rendue plus flexible. Ainsi le séparateur peut être défini par l'utilisateur, telles que la tabulation, la virgule, l'espace, … Le contenu d'un fichier météo de l'application du CIRAD se présente sous cette forme :

Contenu du fichier BassinPlat2008_08_21.txt

```
Date Time	High-Res Temp (*C) c:1 2
18/08/08 12:51	21.71
18/08/08 13:06	22.93
18/08/08 13:21	22.6
18/08/08 13:36	21.83
```

On peut voir que ce fichier est séparé par des tabulations, qu'il contient une première ligne d'entête pour permettre d'identifier le contenu. Chaque enregistrement est composé de deux propriétés : la date de l'enregistrement, et la valeur de température relevée à cette date.

Mais comment lire ces informations depuis une application C# ?

Lecture des fichiers

Il est possible de lire le contenu de fichiers en utilisant les flux de données permis par le Framework .NET. Voici un exemple :

```
StreamReader lecteur = new StreamReader(nomDuFichier);
```

Cette ligne permet l'ouverture du fichier dénommé par la chaine « nomDuFichier » dans un objet en mémoire nommé « lecteur ». C'est cet objet qui permet l'accès aux fichiers, mais aussi à d'autres sources (tout type de flux).
L'utilisation d'une boucle permettra de parcourir le contenu du fichier :

```
while (!lecteur.EndOfStream)
{
   string uneLigne = lecteur.ReadLine();
}
```

L'algorithme est le suivant : On effectue une boucle sans nombre déterminé d'occurrences, avec pour condition de continuer tant que le flux n'arrive pas à la fin de la source (le fichier). Pour chaque exécution de la boucle, on lit le contenu d'une ligne du fichier, et on le place dans la chaine « uneLigne ».

A la fin du fichier, en quittant la boucle il est nécessaire de fermer l'accès au fichier (cette opération est importante et ne doit pas être oubliée, car aucune autre application ne peut accéder à un fichier quand il est ouvert en mémoire !). La ligne suivante permet de libérer le fichier et les ressources mémoire associées :

```
lecteur.Close();
```

Cet ensemble de lignes permet donc d'ouvrir le fichier, de lire chacune des lignes du fichier et de libérer les ressources à la fin.

Le traitement du contenu du fichier est beaucoup plus complexe, il faut pour chaque ligne qui n'est en mémoire qu'une simple chaine de caractères, identifier le type de séparateur employé (ou bien poser la question à l'utilisateur), puis récupérer les données en prenant bien les bonnes informations dans les bonnes colonnes.
L'avantage de l'orienté-objet est que justement il est possible de créer un objet dont le type correspondra avec le contenu générique qu'on trouvera dans un type de fichier, puis de développer une classe qui traitera les lignes et découpera chaque chaine pour en faire un objet. Au final, en passant le nom d'un fichier à une méthode, elle renverra une liste d'objets bien rangés. Ce code étant

très long et complexe, il ne sera pas détaillé dans son intégralité ici.
Il est cependant instructif d'avoir un aperçu du type de commandes employées pour arriver à ce résultat.
Voici donc deux lignes de code qui permettent de séparer une chaine en un tableau de chaines, selon un caractère séparateur défini :

```
char[] separateurs = { ';' };
string[] elements = ligneOriginale.Split(separateurs);
```

La première ligne permet de spécifier un ou plusieurs caractères qui vont servir pour le découpage de la chaine. La deuxième ligne va découper la chaine “ligneOriginale” et remplir le tableau de chaines nommé “elements” avec chaque colonne de données.
On peut ensuite récupérer les données des colonnes en lisant la bonne case du tableau :

```
string dateReleve = elements[2];
```

Le tableau étant d'indice de départ 0, c'est donc la 3ème case qui sera lue pour récupérer la date du relevé.
La création d'une classe pour un type associé au fichier et le remplissage des propriétés de l'objet avec les valeurs relevées dans le fichier permet un transport simple des informations dans l'application.

Écriture dans les fichiers

Dans d'autres cas il est intéressant de pouvoir exporter les données de l'application pour que l'utilisateur puisse en faire autre chose que ce dont l'application est capable. A ce moment là interviennent les flux de données en mode écriture.
La syntaxe à utiliser pour déclarer un objet permettant l'écriture est la suivante :

```
StreamWriter writer = new StreamWriter(nomDuFichier);
```

Cette déclaration va créer un objet en mémoire qui donnera un accès en écriture au fichier défini par “nomDuFichier”. Certains paramètres optionnels existent et permettent de préciser le mode

d'ouverture, ou encore la façon dont seront encodés les caractères dans le fichier.
L'exemple suivant ouvre le même fichier, mais le paramètre défini à False permet de spécifier qu'il ne faudra pas écrire à la suite dans le fichier s'il existe déjà, mais plutôt de remplacer son contenu. Enfin le jeu de caractères Unicode est spécifié pour éviter les défauts d'accents sous Windows.

```
StreamWriter writer = new StreamWriter(nomDuFichier, false, Encoding.Unicode);
```

Les instructions pour écrire sont aussi simples :

```
writer.WriteLine(texte);
```

L'utilisation de cette ligne permet l'envoi des données contenues dans la chaine nommée “texte” dans le fichier sur le disque. Les données sont donc envoyées les unes à la suite des autres, et le fichier est ainsi rempli du début vers la fin. Certaines méthodes (qui ne seront pas décrites ici car non utilisées dans le projet) permettent d'écrire à un endroit précis du fichier, ou d'écrire en données non lisibles (binaire). Ceci permet de créer des fichiers pour stocker d'autres types d'informations, telles que les images, les fichiers audio, vidéo, etc.
Il ne faut pas oublier, comme pour la lecture, de fermer le fichier à la fin des opérations.

```
writer.Close();
```

Le fichier est donc fermé, les données envoyées sont donc bien validées et transférées sur le disque, dans les cas où cela n'aurait pas déjà été fait (écriture différée gérée par le système).

Pour écrire un fichier à partir d'une liste d'objets, un code de ce type conviendrait :

```
anagmaxDBEntities monAcces = new anagmaxDBEntities();
List<Variete> listeVarietes = monAcces.Variete.ToList();
StreamWriter writer = new StreamWriter(“sortie.csv”, false, Encoding.Unicode);
foreach(Variete v in listeVarietes)
{
        writer.WriteLine(v.ID + “\t” + v.nom + “\n”);
}
writer.Close();
```

Ce code accède à la base de données, récupère la liste des variétés, ouvre un fichier “sortie.csv” en écriture, remplacement et Unicode, et écrit pour chaque variété le ID de la variété, suivi d'une tabulation, suivi du nom de la variété, et terminé par un retour à la ligne. Le fichier est à la fin fermé et les ressources sont libérées. Nul besoin de plus de code pour exporter des informations dans un fichier.
Le cadre dynamique de l'application rendra nécessaire l'utilisation d'une fenêtre pour sélectionner le fichier à enregistrer, et le nombre de paramètre à sauvegarder pourra changer.

Une fois la lecture et l'écriture dans les fichiers possibles, il est tout aussi nécessaire de pouvoir sortir des données dans le but qu'elles soient imprimées... Et là, les fichiers CSV ne suffisent plus.

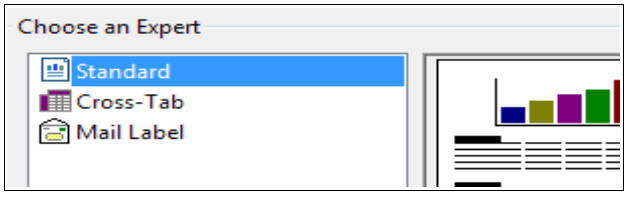
Rapports de données

Les rapports de données permettent de mettre en page des informations pour que l'utilisateur puisse directement les lire et les imprimer. Il faut donc que la mise en page soit correcte, que les données soient ordonnées, voire même que des emplacement soient prévus pour que cet utilisateur puisse y écrire des annotations.

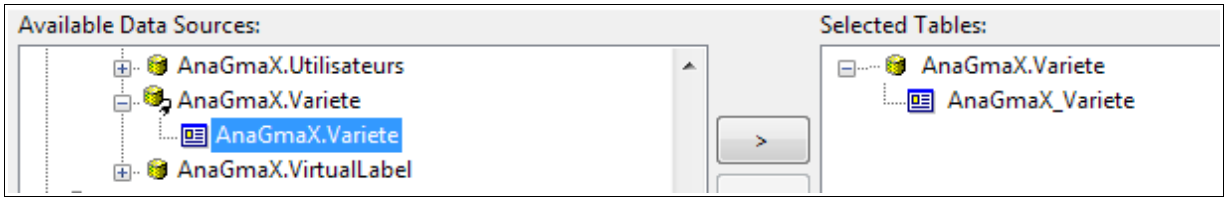
Microsoft Visual Studio a ainsi intégré dans sa version PRO un outil dédié à cette tache, nommé Crystal Reports. Cet outil développé par Business Objects (entreprise qui a été rachetée amicalement en 2008 par SAP) permet la création de rapports génériques (contenant des données textuelles, graphiques et des images) et ces rapports peuvent ensuite être remplis dynamiquement pendant l'exécution du logiciel.

La création de rapport Crystal Reports est aussi guidée par un assistant.

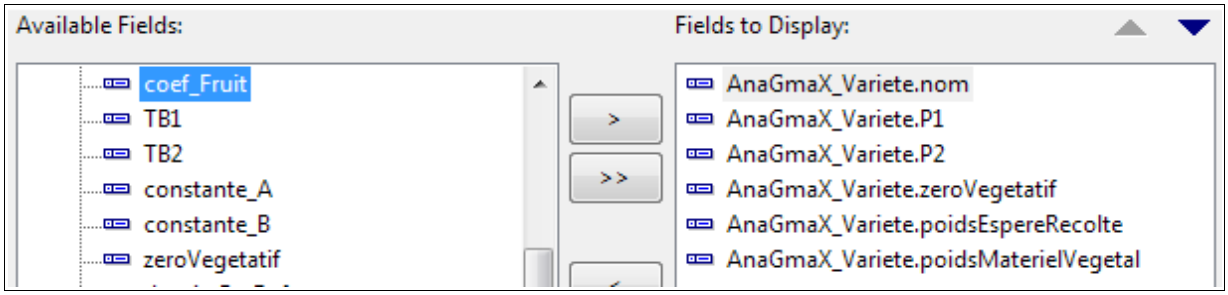
Il faut d'abord choisir si on crée un rapport ou si on en ouvre un existant, et le type de rapport si nouveau :



Une fois cette étape passée il vous sera demandé à partir de quel(s) objet(s) vous souhaitez créer un rapport. Cette étape est décisive puisque le rapport créé automatiquement le sera en fonction des réponses de cette étape. Dans nos démonstrations nous continuons avec le modèle Variété :

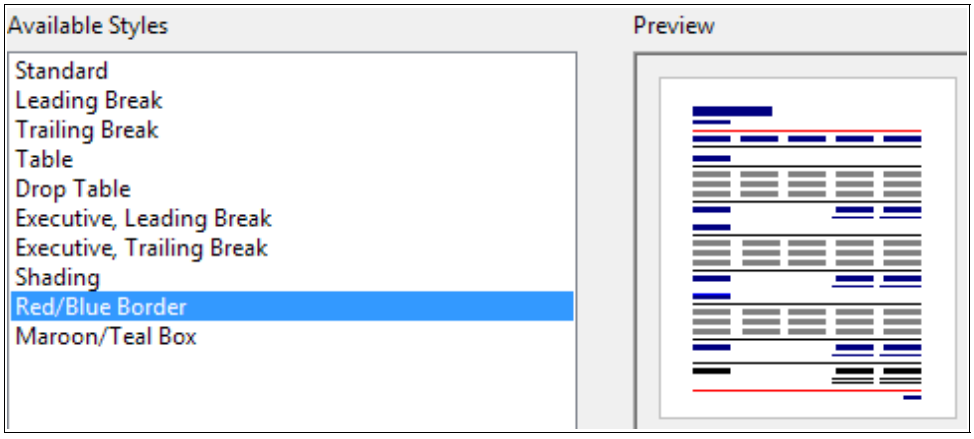


Ensuite il faut déterminer les champs qui seront affichés dans le rapport. Libre choix au créateur de montrer ce qui qui lui semble pertinent :

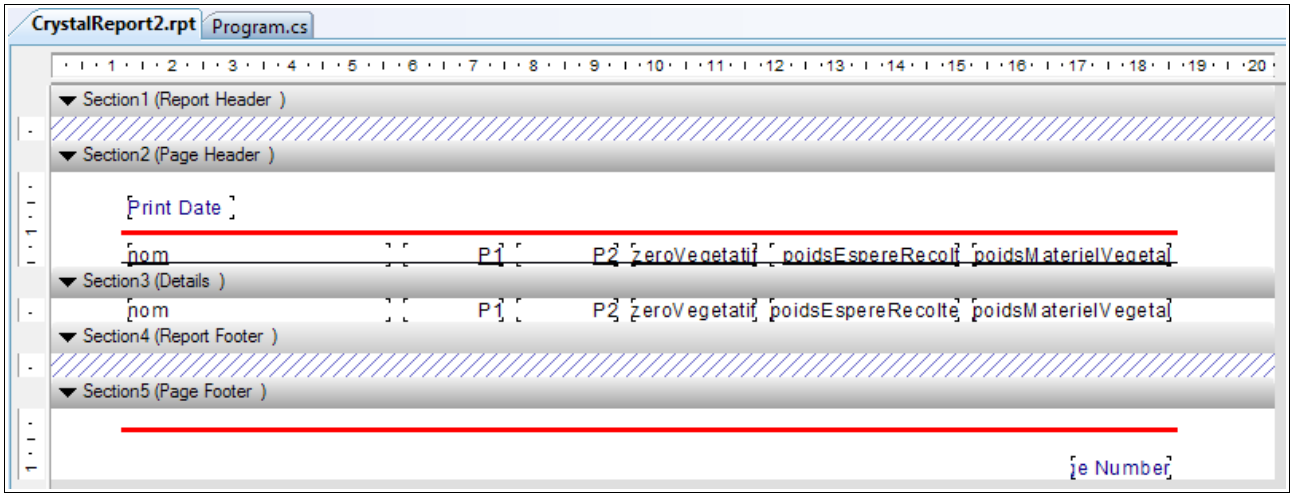


On peut ensuite choisir si le formulaire doit regrouper les éléments selon certains champs...
On peut aussi déterminer certains types de filtres sur le contenu à afficher.

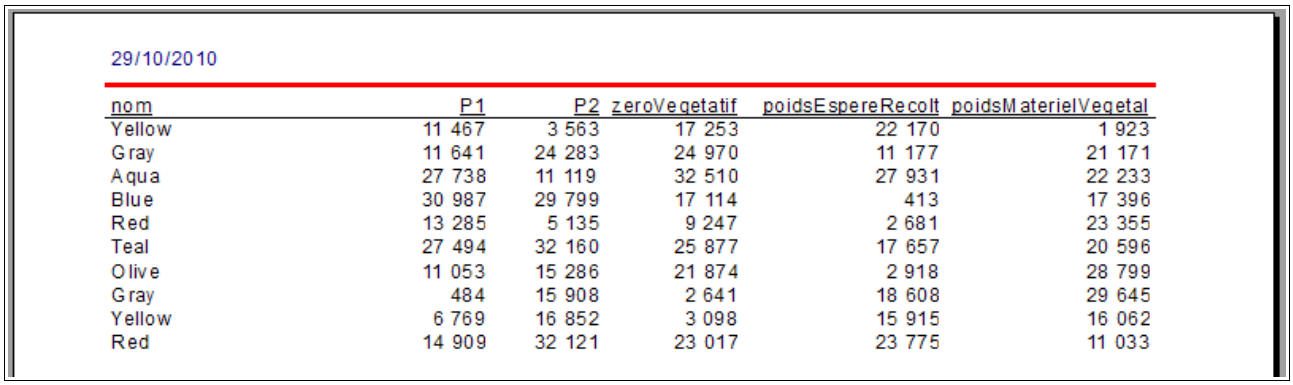
Vient le moment de choisir le style visuel :



Le rapport est ensuite terminé, l'éditeur s'ouvre et affiche le rapport en mode « modifiable ».



Les modifications dans l'éditeur sont délicates, de nombreux essais sont nécessaires pour arriver au résultat, mais sans trop d'expérience et avec un peu de patience, le résultat est là : on obtient un beau rapport avec très peu de manipulations :



29/10/2010

nom	P1	P2	zeroVegetatif	poidsEspereRecolt	poidsMaterielVegetal
Yellow	11 467	3 563	17 253	22 170	1 923
Gray	11 641	24 283	24 970	11 177	21 171
Aqua	27 738	11 119	32 510	27 931	22 233
Blue	30 987	29 799	17 114	413	17 396
Red	13 285	5 135	9 247	2 681	23 355
Teal	27 494	32 160	25 877	17 657	20 596
Olive	11 053	15 286	21 874	2 918	28 799
Gray	484	15 908	2 641	18 608	29 645
Yellow	6 769	16 852	3 098	15 915	16 062
Red	14 909	32 121	23 017	23 775	11 033

Coté code, la création et l'affichage de rapport est aussi très simple :

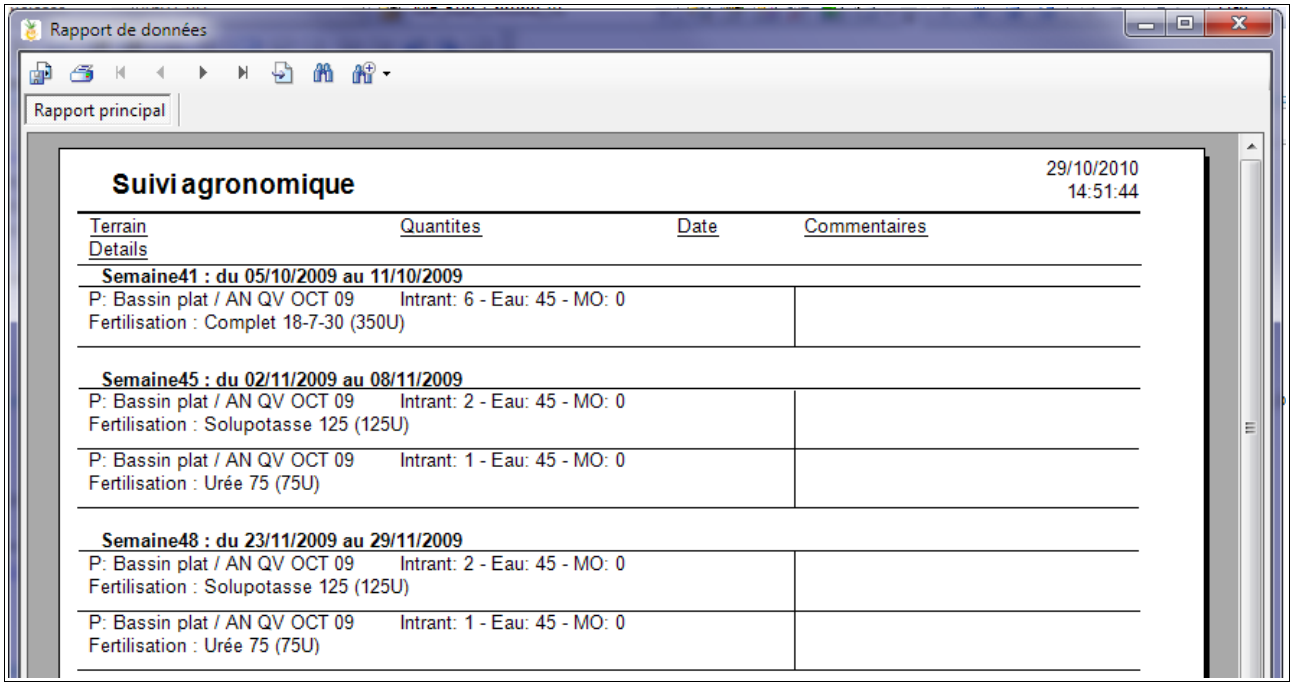
```
CrystalReport2 rapport = new CrystalReport2();
rapport.SetDataSource(maListeDeVarietes);
crystalReportViewer1.ReportSource = rapport;
```

Explications concernant ces lignes de code :

- La première ligne permet la création d'un objet en mémoire à partir du modèle créé par l'assistant.
- La deuxième ligne remplit le rapport avec les données de notre liste de variétés, qu'elles viennent de la mémoire ou de la base de données.

- La troisième ligne remplit un contrôle qu'on aura auparavant disposé sur la fenêtre avec l'éditeur visuel. C'est ce contrôle qui sera « l'écran » d'affichage de notre rapport.

Voici un aperçu d'un vrai rapport du logiciel du CIRAD, en version finale après le stage :



Ce document est immédiatement lisible, enregistrable (formats PDF, XLS, DOC, RTF, RPT...) et imprimable.

Les utilisateurs peuvent donc sur le logiciel du CIRAD consulter les taches à faire sur le terrain, remplir leurs feuilles et revenir saisir les taches réalisées sur le logiciel.

Les classeurs Excel qui faisaient le même travail offraient beaucoup moins de performance, n'étaient pas dynamiques, ou du moins ne permettaient pas de consulter instantanément les listes à faire...

Démonstration de l'originalité de l'approche

Ma présence au CIRAD pouvait être comparée à la présence d'un prestataire de services extérieur, puisque j'ai dû avoir une méthode de travail réfléchie, ordonnée, sans compter la gestion du travail en équipe sur la première moitié du développement. Ainsi il m'a été offert la possibilité d'apprendre à rendre des travaux et des services de la qualité la plus professionnelle possible.

Règles de programmation

Une des choses les plus importantes du projet a été de rendre l'application terminée, fonctionnelle, dans les temps. Ceci particulièrement parce que la version précédente tentée en VB.NET par un autre étudiant n'avait pas pu être terminée. L'objectif a pu être atteint avec un peu de retard sur les prévisions, mais avec plus d'améliorations, de performances et de simplicité d'utilisation que le projet initial.

Il a par contre été important de programmer selon 3 règles fondamentales :

- Commentaires le code
- Nomenclature : noms de variables et de méthodes
- Structuration et clarté du code

Commentaires du code

Il est d'usage de commenter chaque ligne de code (ou portion de code) en plaçant le commentaire avant la ligne concernée. Dans certains cas la ligne étant très courte (déclaration de variables), le commentaire peut être placé à la fin de la ligne :

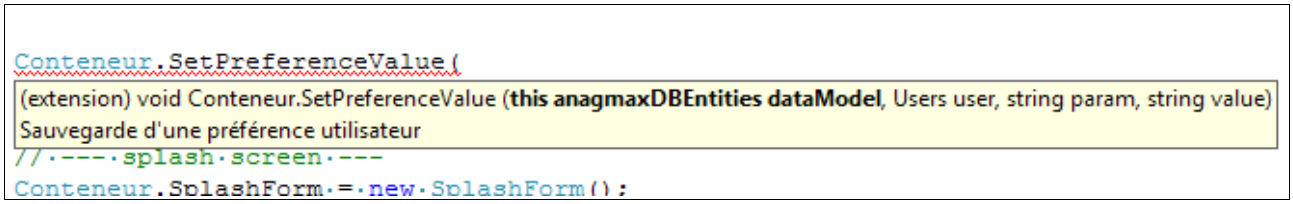
```
// --- chargement des préférences ---
Conteneur.DataModel.ChargePrefs();
// --- page principale ---
Conteneur.MainForm = new FormParente();
// --- lancement de l'application ---
Application.Run(Conteneur.MainForm);
```

Pour chaque méthode rédigée, il est aussi important de décrire la méthode, ses paramètres et ce qu'elle renvoie.

Ci-dessous un aperçu de la structure du commentaire à mettre devant chaque méthode :

```
/// <summary>
/// Sauvegarde d'une préférence utilisateur
/// </summary>
/// <param name="prop">utilisateur propriétaire de la préférence</param>
/// <param name="param">nom du paramètre à mémoriser</param>
/// <param name="value">valeur à mémoriser pour le paramètre</param>
public void SetPreferenceValue(this anagmaxDBEntities dataModel, Users user, string param, string value)
```

Ces informations permettent lors de la relecture de bien comprendre chaque élément, et permettent aussi à l'auto-complétion intégrée dans Visual Studio de proposer une description complète de la méthode lorsque l'on s'en sert. Voici ce que propose Visual Studio pour cette méthode :



Cela n'a pas été fait sur notre projet, mais certains logiciels, comme OxyGen Code Generator, peuvent récupérer l'ensemble des commentaires, les mettre en page pour faire une documentation complète du code de l'application. C'est une pratique très utile pour fournir un produit clé en main destiné à être modifié par la suite. Mais elle demande un temps considérable pour commenter vraiment toute l'application.

Nomenclature dans le projet

De la même façon qu'il est important de commenter le code afin qu'il soit plus compréhensible à la relecture, il est aussi important d'utiliser des noms explicites et ayant un sens dans la rédaction du code. Démonstration par un exemple.

Ce code :

```
double densite = StaticCalculsTerrain.CalculDensite((billon as Terrain).GetParcelleParente());
double poidsTIF = billon.poidsEspereAuTif;
double varA = billon.GetParcelleParente().Variete.constante_A;
double varB = billon.GetParcelleParente().Variete.constante_B;
double sommeTemp = (poidsTIF - poidsMatVeg - varB) / ( densite * coef_Plant + varA);
```

est beaucoup plus lisible et compréhensible que cet autre code, pourtant réalisant exactement la même chose :

```
double d = SCTerrain.C_Densite((b as Terrain).GetPaParent());
double pTIF = billon.pTif;
double a = b.GetPaParent ().var.cA;
double b = b.GetPaParent ().var.cB;
double sTemp = (pTIF - pMatV - b) / ( d * coef_Plant + a);
```

C'est pourquoi il est vraiment important de suivre ces conseils :

- Utiliser des noms explicites pour les objets.
- Éviter les noms subjectifs, qui n'ont pas de sens profond dans le contexte.
- Ne pas hésiter à concaténer les noms dans une chaine : GetParcelleParente()
- Garder toujours la même façon de nommer les éléments dans l'ensemble de l'application :
  - Casse Pascal : La première lettre de l'identificateur et la première lettre de chaque mot concaténé suivant sont en majuscules. Vous pouvez utiliser la casse Pascal pour les identificateurs comportant trois caractères ou plus.
  - Casse mixte: La première lettre de l'identificateur est en minuscules et la première lettre de chaque mot concaténé suivant est en majuscules.
  - Majuscules : Toutes les lettres de l'identificateur sont en majuscules.

Microsoft recommande l'utilisation de ces règles pour le nommage :

Identificateur	Casse	Exemple
Classe	Pascal	AppDomain
Type énumération	Pascal	ErrorLevel
Valeurs d'énumération	Pascal	FatalError
Événement	Pascal	ValueChanged
Classe d'exceptions	Pascal	WebException
Champ statique en lecture seule	Pascal	RedValue
Interface	Pascal	IDisposable
Méthode	Pascal	ToString
Espace de noms	Pascal	System.Drawing
Paramètre	Casse mixte	typeName
Propriété	Pascal	BackColor

Cela n'est pas vital, mais son utilisation permet d'écrire du code qui soit universel, lisible, et permet d'identifier les éléments par la façon dont ils sont écrits...

Clarté du code

Le code avec des variables bien nommées et des commentaires ne suffit parfois pas pour une bonne lecture.
Il est alors autorisé et recommandé d'utiliser de petites astuces :

- ne pas hésiter à laisser une ligne vide pour bien séparer des blocs.
- Éviter de mettre plusieurs instructions sur seule ligne (car cela est possible).
- Toujours bien utiliser les indentations de code (Visual Studio s'en occupe pour vous, pour la modique somme d'un raccourcis clavier : Ctrl+E suivi de D).
- Les blocs délimités par les accolades ouvrantes {et fermantes } sont importantes lors d'utilisation des if, while, do, ainsi que les blocs des méthodes et autres non cités... Mais elle peuvent être utilisées partout, pour quelque raison que ce soit. Elles forcent l'indentation, et permettent de bien séparer le code.

Voici un exemple de code avec des séparations :

```
            Thread.CurrentThread.CurrentCulture = ci;
            Thread.CurrentThread.CurrentUICulture = ci;
        }

        //Language.LoadLanguage("fr");
        Help.LoadHelp(langue);

        // locales pour les messagebox
        MessageBoxManager.OK = global::AnaGmaX.Properties.Resources.ok;
        MessageBoxManager.Cancel = global::AnaGmaX.Properties.Resources.Annuler;
        MessageBoxManager.Retry = global::AnaGmaX.Properties.Resources.reessayer;
        MessageBoxManager.Ignore = global::AnaGmaX.Properties.Resources.ignorer;
        MessageBoxManager.Abort = global::AnaGmaX.Properties.Resources.Annuler;
        MessageBoxManager.Yes = global::AnaGmaX.Properties.Resources.oui.UCFirst();
        MessageBoxManager.No = global::AnaGmaX.Properties.Resources.non.UCFirst();
        MessageBoxManager.Register();

        MFControls.VeriSecur.Load();

        Conteneur.CancelOperation = false;

        Conteneur.TitleVersion = global::AnaGmaX.Properties.Resources.Anagmax + " v" + Conteneur.NumVersion;

        // ---- effets de fenetre ---
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);

        // ---- splash screen ---
        Conteneur.SplashForm = new SplashForm();
        Conteneur.SplashForm.Show();
        Application.DoEvents();

        if (Process.GetProcessesByName(Process.GetCurrentProcess().ProcessName).Count() > 1)
            '
```

Optimisation du code

Lorsque l'application a passé l'étape principale du développement (elle démarre, fait ce pourquoi elle a été créée, et ne présente plus de problèmes et de bogues), il reste une étape importante qu'est l'optimisation.
Pendant cette phase il faut réduire le temps d'exécution des méthodes les plus gourmandes (qui sont déterminées par l'utilisation de la version fonctionnelle du logiciel). Certains autres codes ne peuvent pas être améliorés concernant le temps, mais peuvent parfois être réécrits en un code beaucoup moins long.
C'est en général les méthodes principales des grosses opérations qui peuvent être modifiées, parfois en les séparant en plusieurs méthodes, parfois en les réécrivant entièrement. Car dans bien des cas, la manière de penser et de résoudre un problème n'est pas la même au début du projet, et à la fin du projet. L'expérience acquise sur un même projet permet de proposer des solutions radicalement différentes concernant un même problème, en prenant en compte les problèmes rencontrés dans les premières versions, et les optimisations apportées ensuite.

La programmation orienté-objet permet de déjà bien délimiter les actions et les éléments, ce qui rend le code plus logique et plus simple (notamment en passant des objets en paramètres plutôt que des listes de paramètres...).

Les boucles sont à vérifier en fin de développement, car souvent un objet créé en cours de route ou une méthode ajoutée peut simplifier voire remplacer une boucle qui a été écrite depuis le début du projet.

Une bonne sélection des types de variables contribue aussi à la bonne optimisation de l'application. Il est nécessaire de bien définir les types à utiliser, depuis la création de la base de données, jusqu'aux dernières lignes de codes.
Par exemple s'il est possible d'utiliser un INT pour une variable, cela vaut mieux que d'y laisser un LONG, qui consomme plus de mémoire.
Il faut éviter un maximum les conversions inutiles : une méthode prenant un entier en paramètre, faisant un calcul d'entiers, et renvoyant des entiers, est mieux conçue qu'une qui reçoit et envoie des entiers et fait les traitements internes avec des entiers longs...

L'utilisation d'un type simple (entier, long, réel, etc.) est plus rapide que la création et l'utilisation d'un objet, si celui ci n'apporte aucun avantage propre aux objets...

D'autres règles à savoir :

- lors de l'utilisation de tables, essayer de toujours avoir un minimum de dimensions possible.
- L'utilisation d'une table de tables est plus performante que d'utiliser des tables multidimensionnelles.
- Faire une recherche dans une tables est plus rapide que de chercher dans une collection.
- Accéder à une variable est plus rapide que d'accéder à un élément d'une table.
- Donc attribuer une valeur d'un tableau à une variable s'il faut l'utiliser de nombreuses fois.
- L'utilisation de List ou de Collection est recommandée si le nombre d'éléments est inconnu.
- Éviter d'utiliser des types inconnus (déclaré var et qui prend le bon type lors de l'exécution) et Object. En effet la recherche du bon type pendant l'exécution prend du temps.

Lors de l'utilisation de Switch, ainsi que la gestion des erreurs Catch, il est conseillé de placer les blocs les plus utilisés en premier, pour éviter de parcourir les conditions inutilement à la recherche de la bonne situation.

Quelques astuces sur les opérateurs :

- Utiliser les ET (&&) et les OU (||) plutôt que d'imbriquer plusieurs conditions IF.
- Un incrément du type i++; est plus performant que i = i + 1;
- Un calcul du type a += 4; est plus performant que a = a + 4;
- Optimiser les calculs en utilisant des opérateurs moins gourmands pour le processeur :
  - les additions sont plus rapides que les multiplications.
  - Log, Sinus et Cosinus sont très gourmands en ressources.
  - Stocker le résultat d'un calcul plutôt que de le calculer plusieurs fois.
- Limiter les conversions et optimiser celles qui sont nécessaires.

Il est possible d'optimiser une boucle. En effet, commencer par mettre toutes les opérations qui ne sont pas modifiées avant la boucle (déclaration d'une variable, attribution d'une valeur fixe, etc.), permet un gain de temps et évite des opérations mémoires redondantes dans la boucle.

A des fins de fiabiliser l'application, il faut essayer de prendre en compte toutes les erreurs possibles (utilisation des Try... Catch...) afin que l'application ne se retrouve pas dans une situation de perte de données (vital pour l'utilisateur).

Pour limiter l'exécution de lignes non désirées dans des cas précis, ne pas hésiter à quitter la méthode (return).

Pour pouvoir satisfaire la demande utilisateur il est parfois utile d'exécuter les calculs ou des taches en arrière plan (avec des threads).

Il est même possible de gagner du temps sur les chaines de caractères, notamment en utilisant la classe StringBuilder plutôt que de faire de multiples concaténations.

L'affichage peut aussi être optimisé : il est préférable de cacher la fenêtre pendant la réactualisation de ses contrôles, et de l'afficher ensuite, plutôt que de successivement changer les valeurs de chaque contrôle devant l'utilisateur. Il en est de même pour les listes : une longue liste cachée pendant son remplissage est préférable.

Concernant les fichiers, il est préférable de traiter les données en mémoire plutôt que sur le fichier directement. Il est donc plus judicieux de charger le fichier en premier temps, puis traiter les données.

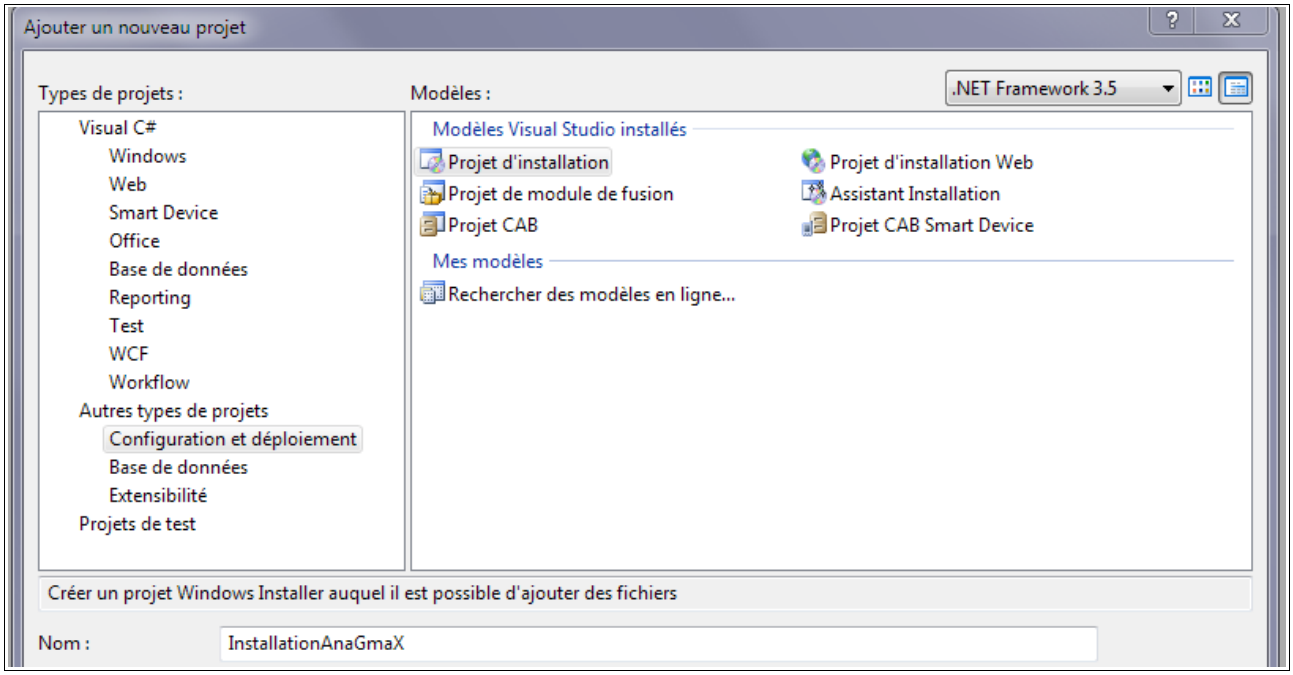
Déploiement

Une fois l'application développée, optimisée, fiabilisée, il faut pouvoir la distribuer aux utilisateurs de façon à ce qu'elle soit facile à mettre en place.

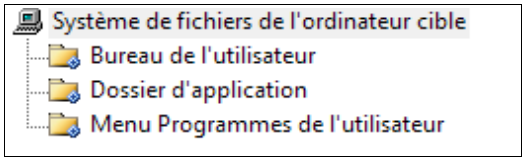
C'est le rôle d'une application de déploiement « Microsoft installer ».

Visual Studio est capable de générer de manière (presque) automatisée un programme d'installation en .EXE ou .MSI.

Toujours avec l'assistant, on crée un nouveau projet d'installation :



Il suffit de valider pour que le projet soit terminé. Il faut cependant le paramétrer un peu pour qu'il fonctionne. Une liste des dossiers systèmes qu'utilisera l'installation est alors affichée :

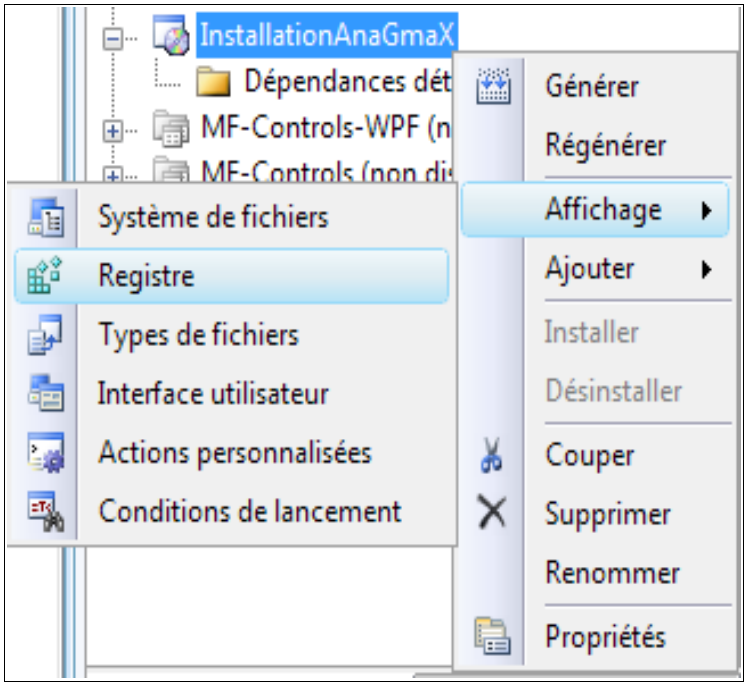


- Le bureau de l'utilisateur : destiné aux fichiers qui seront copiés sur le bureau (icône dans notre cas).
- Le dossier d'application : le dossier du logiciel dans « Program Files ». Par exemple CIRAD
- Le menu Programmes : le menu démarrer de l'utilisateur, où seront mises des icônes également.

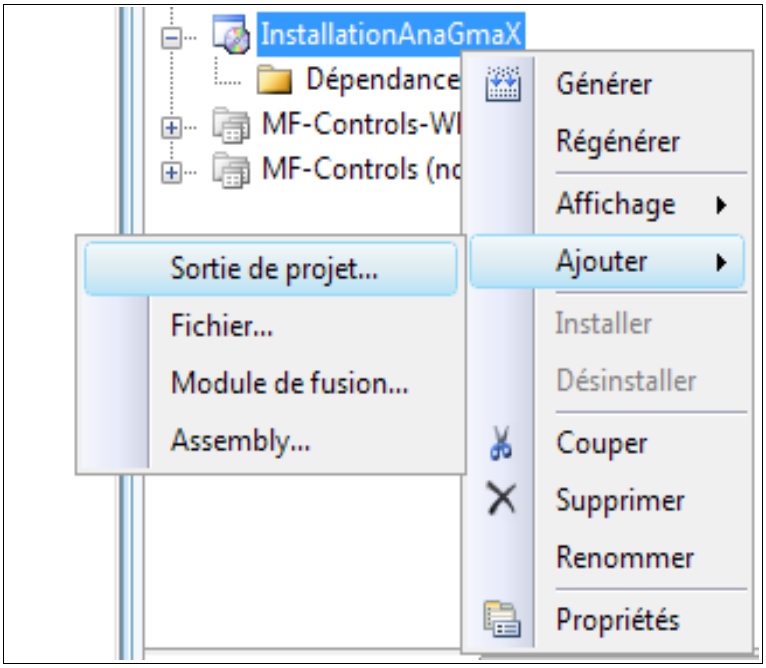
A cette liste de dossiers peuvent être rajoutés tous les dossiers désirés.

Le projet d'installation permet de personnaliser tous les éléments de l'utilisateur : édition du registre, création de fichiers et de dossiers, style du programme d'installation, nom, icône, titre, auteurs...

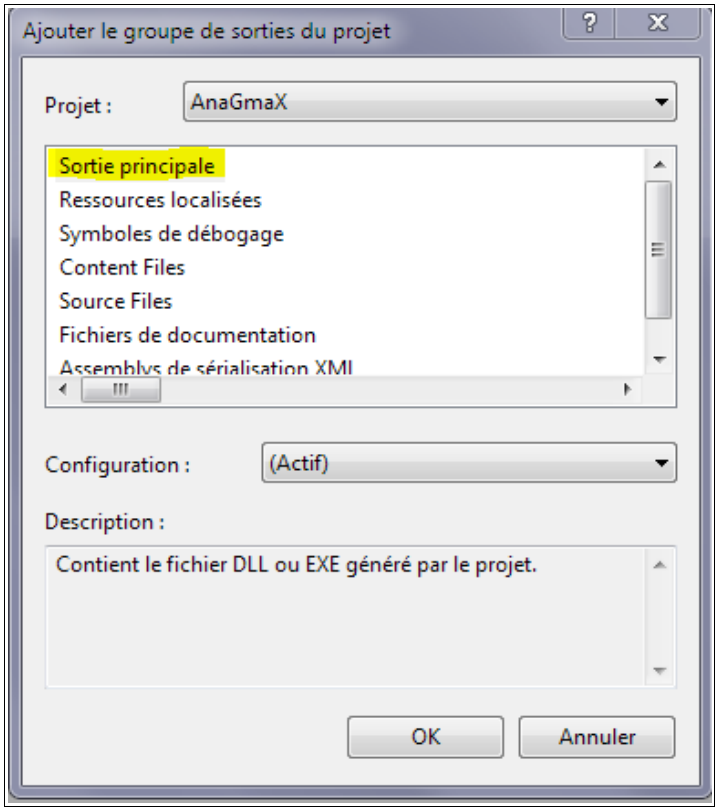
L'accès aux différentes sections est possible depuis un bouton droit sur le projet d'installation > Affichage.



A ce projet il manque une chose importante : il faut spécifier quel projet doit être déployé. Ce projet nommé « sortie principale » doit être ajouté à la main. Bouton droit > Ajouter > Sortie de projet.



On voit donc apparaître la liste des projets, et pour le projet concerné on doit sélectionner la sortie principale (= le .EXE de l'application finalement).



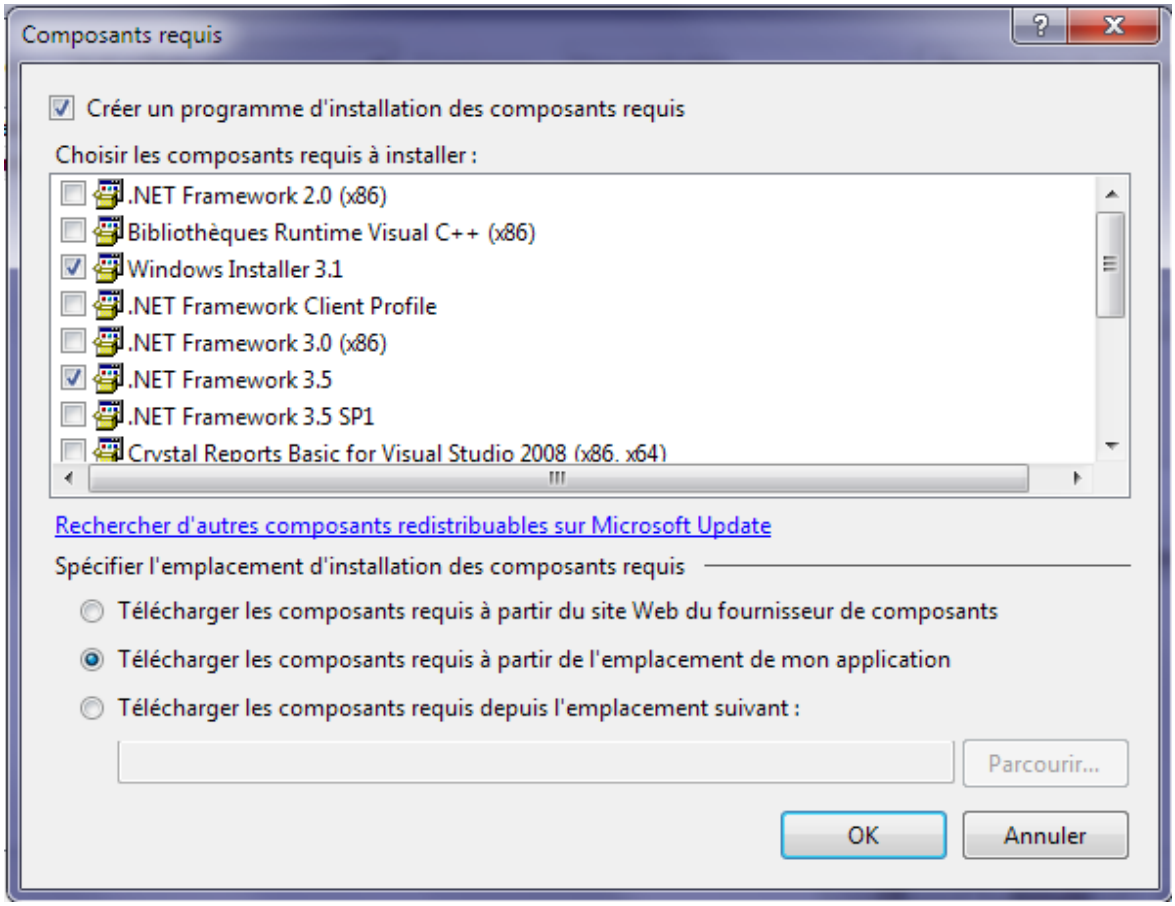
On peut aussi choisir d'autres sorties : les ressources, les fichiers source du projet, la documentation...

La sortie principale apparaît dans la liste des fichiers à copier sous le Dossier d'application.
Une fois les raccourcis paramétrés, les options de l'installation définies, il ne reste qu'à générer le programme d'installation : Bouton droit sur le projet d'installation > Générer.

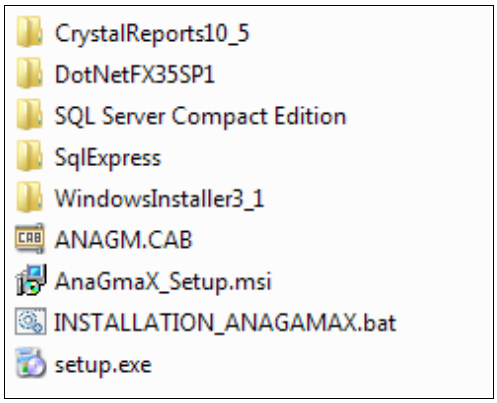
On peut finalement retrouver le projet d'installation exécutable dans le dossier Debug ou Release (selon état de compilation).

Notez qu'il est possible d'inclure les dépendances (Framework, SQL Server, …) dans le programme d'installation. Bouton Droit sur le projet d'installation > Propriétés. Un bouton Composants requis donne accès à ces choix.

A ce moment précis il faut choisir les éventuelles références qui n'auraient pas été automatiquement détectées, et choisir l'option de mettre les composants dans le dossier de l'installation (ou bien choisir de les installer depuis Internet lors de l'installation : choix par défaut).



L'application complète AnaGmaX (nom final donné au projet Oumpapa du CIRAD) se présente ainsi :



Le tout tient sur un total de 366 mégaoctets et s'installe sur toute plateforme Windows depuis XP.

Conditions d'application de l'approche présentée

Mon rôle sur ce projet fut très similaire à un prestataire de services, j'ai dû faire un énorme travail de recherche et de compréhension sur le métier existant, autant sur la culture d'ananas que sur les technologies à utiliser.
Cela a donc rendu plus facile la rédaction de ce mémoire, et l'apport en documentations et en concepts.

Tout le travail fourni pour terminer cette application, ainsi que les parties de codes récupérées de l'ancienne version (calculs de Patrick Fournier et de toute son équipe au CIRAD), m'ont fait prendre conscience de l'énorme travail qui avait déjà été réalisé au CIRAD. J'ai aussi compris l'importance qu'avait eu le travail que j'ai réalisé chez eux.
C'est pourquoi j'ai décidé de présenter ce rapport de cette façon, à savoir présenter les outils avec lesquels j'ai pu terminer un long projet, sur plusieurs stages, avec même un début en équipe.

Ce plan ne se limite donc pas au simple projet du CIRAD, mais reprend de manière générale la technique d'approche et les outils à connaitre pour réaliser de puissantes applications en C#.NET.

Conclusion

Au bout de deux ans de développement sur le logiciel AnaGmaX, dont une année en collaboration avec Mickael FOLIO, j'ai pu terminer le logiciel dans une version fonctionnelle, rapide, et surtout conviviale. J'ai pu participer à la formation des utilisateurs, j'ai pu aussi pendant la phase de développement, participer à des réunions avec eux, pour encore mieux cerner leurs besoins, leurs attentes.

Leurs critiques ont aussi et bien entendu permis de faire avancer les choses.

Le travail en continu avec mon responsable de stage a permis de suivre à la fois ses demandes, son « cahier des charges », mais m'a aussi permis d'imposer mes idées, qui ont fait radicalement changer le projet de route, à plusieurs reprises. Les points de vues se complètent toujours, il est important de les partager.

Le C# était pour moi un langage connu, mais toutes les autres technologies utilisées étaient nouvelles. Et ce qu'elles apportent aux projets m'a permis de considérablement améliorer mon niveau, confirmant ma préférence et ma spécialité en tant que développeur.

Une fois l'application étant reconnue finie et distribuée chez les utilisateurs, une autre grande surprise m'attendait: cela a été la demande du responsable de stage à ce que le CIRAD dépose un brevet pour l'application. C'était la reconnaissance finale que mon travail avait été un franc succès.

La dernière étape optionnelle suggérée par un des utilisateurs : porter l'application sous Linux avec Mono... J'ai commencé à me poser la question, mais Entity sous Linux, ce n'est pas pour aujourd'hui...

Références

- Mes connaissance pour le langage C#
- http://www.msdn.fr pour les trous de mémoire
- http://www.osnews.com/story/5602/Nine_Language_Performance_Round-up_Benchmarking_Math_File_I_O/page3/ pour les performances
- http://translate.google.fr pour quelques compréhensions
- http://www.cirad.fr l'entreprise et les chiffres